UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

GRADO EN INGENIERÍA INFORMÁTICA

-TRABAJO FIN DE GRADO-

SIMULADOR VIRTUAL PARA SISTEMAS MULTI-CÁMARA DISTRIBUIDOS

Luis Pérez Llorente
Tutor: Juan Carlos San Miguel
Ponente: José María Martínez Sánchez

Julio 2015

SIMULADOR VIRTUAL PARA SISTEMAS MULTI-CÁMARA DISTRIBUIDOS

Autor: Luis Pérez Llorente
Tutor: Juan Carlos San Miguel Avedillo
email: {luis.perezllorente@estudiante.uam.es, juancarlos.sanmiguel@uam.es }

Video Processing and Understanding Lab
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio 2015

Resumen

Este trabajo Fin de Grado presenta un sistema capaz de simular el manejo de múltiples cámaras capturando información visual de entornos virtuales 3D. Este sistema permite definir entornos de pruebas para investigar en algoritmos de vision por computador (*Computer Vision*), por ejemplo de seguimiento o detección de personas en vídeo, donde se pueden repetir las situaciones de interés todas las veces que sea necesario. El desarrollo del sistema se basa en la herramienta *Virtual Video*, software creado cpara realizar diferentes operaciones en cámaras situadas en entornos 3D basados en *source SDK* (e.g. Half Life 2). Este TFG incrementa las funcionalidades de la herramienta mediante un control realizado de forma remota siguiendo la metodología cliente-servidor, permitiendo crear, borrar y modificar la posicion y el campo visual de varias cámaras de manera simúltanea. Adicionalmente se proporciona un visionado en tiempo real de imágenes capturadas por las múltiples cámaras instaladas. El entorno virtual donde se colocarán las cámaras simula con gran realismo distinto tipos de escenarios, que pueden ser generados mediante herramientas de diseño de mapas 3D (e.g. *Hammer*). Posteriormente, se realiza un análisis del rendimiento del sistema desarrollado en sistemas distribuidos de múltiples cámaras, considerando los recursos requeridos.

Abstract

The development of a system that is able to simulate the management of multiple cameras taking visual information from 3D virtual environments. This system enables to define test environments in order to research on computer vision algorithms (Computer Vision), for example people tracking or detection algorithms recorded in a video, which allows you to repeat the situation as often as necessary. The development of the system is based on the tool called Virtual Video. This software was created to do performs in cameras from 3D environments that are based on source SDK (v.gr. Half Life 2). The present study increases the number of features that the tool has. This is done by remote control following the client server model letting the user create, cancel and modify the location and the visual field of different cameras simultaneously. In addition, the system shows a real-time viewing with pictures took by the multiple cameras that were installed. The virtual environment in which the cameras will be installed simulate in a very realistic way different kinds of scenarios. These scenarios can be generated with 3D map designing tools (v.gr. Hammer). Afterwards, taking into account the required resources, a perform analysis of the developed system is done using distributed systems multi-camera.

Palabras clave

Sistema multi-cámara, Entornos virtuales 3D, Computer Vision, *Virtual Video*, *Source SDK*.

Agradecimientos

En primer lugar quisiera agradecer a los que más se lo merecen, a mis padres y hermanas, que tanto me aguantado durante el transcurso de esta época de mi vida. Luego agradecer también a mis amigos y compañeros de la Universidad, sin vosotros esto no hubiese sido igual, y por último a mi tutor Juan Carlos, por guiarme y darme la oportunidad trabajar junto al VPU.

¡Gracias!

Dedicatoria

Para mis abuelos, Luis y Antonia, de vuestro nuestro nieto que os quierre mucho.

Luis Pérez Llorente
Julio 2015

Índice general

Índice de figuras

Índice de cuadros

Acronimos

OpenCV Open Source Computer Vision.

WiSE-MNet Wireless Simulation Environment for Multimedia Networks.

CamSim Smart Camera Simulation Environment.

SLCNR Software Laboratory for Camera Networks Research.

WSVN Wireless Sensors & Video Networks.

M3WSN Mobile Multi-Media Sensor Network.

FOV Field Of Vision.

VPU Video Processing and Understanding Lab.

EPS Escuela Politécnica Superior.

UAM Universidad Autónoma de Madrid.

IP Internet Protocol.

TCP Protocolo de Control de Transmisión.

Capítulo 1

Introducción

1.1. Motivación

En los últimos años ha habido una gran evolución tecnológica, lo que ha causado un aumento del desarrollo de nuevos sistemas con cámaras distribuidos para vídeo-vigilancia [2]. Con todo ello, ha surgido la necesidad de generar sistemas que sean capaces de controlar múltiples cámaras conectadas. Estos sistemas deberán ser capaces de motorizar las diferentes cámaras de manera autónoma, teniendo la posibilidad de mover las cámaras (girar, inclinar, zoom..), gestionarlas (crear cámara, eliminar cámara, desconectar cámara...) y extraer información (frame, ground truth...), así pues, múltiples lineas de investigación han aparecido para proporcionar la tecnología necesaria para estas aplicaciones.

La investigación en sistemas multi-cámara distribuidos presenta una gran complejidad y variabilidad (ver Figura 1.1), siendo de especial dificultad la obtención de entornos controlados en los cuales se puedan repetir las situaciones de interés. Esta *repetibilidad* se considera clave para poder evaluar nuevos algoritmos frente a alternativas existentes. Para una misma acción (e.g. abandono de un objeto), las múltiples cámaras pueden moverse de manera distinta dependiendo del algoritmo utilizado, asi pues requeriendo que la acción se repita en todas las pruebas realizadas. En este contexto, aparecen los simuladores de sistemas multi-cámara, que proporcionan datos sinteticos representando una situación real (e.g. obtenidos de un simulador 3D).

Los entornos donde se generarán las simulaciones de estos sistemas deberán de ser capaces de reproducir escenarios con gran semejanza a espacios de la vida real (Estación de metro, rotonda, colegio..), para que a la hora de ejecutar algoritmos, ya sea por ejemplo de seguimiento de personas o de detección de intrusos entre otros, los resultados obtenidos tengan un alto índice de exactitud. Al generar los escenarios de manera virtual por medio de editores de mapas, permite obtener multitud de diferentes ambientes donde simular la integración de las cámaras. Estos escenarios se desarrollarán en un espacio con tres dimensiones.

Además será necesario que el sistema sea capaz de manejar de forma sencilla todo tipo

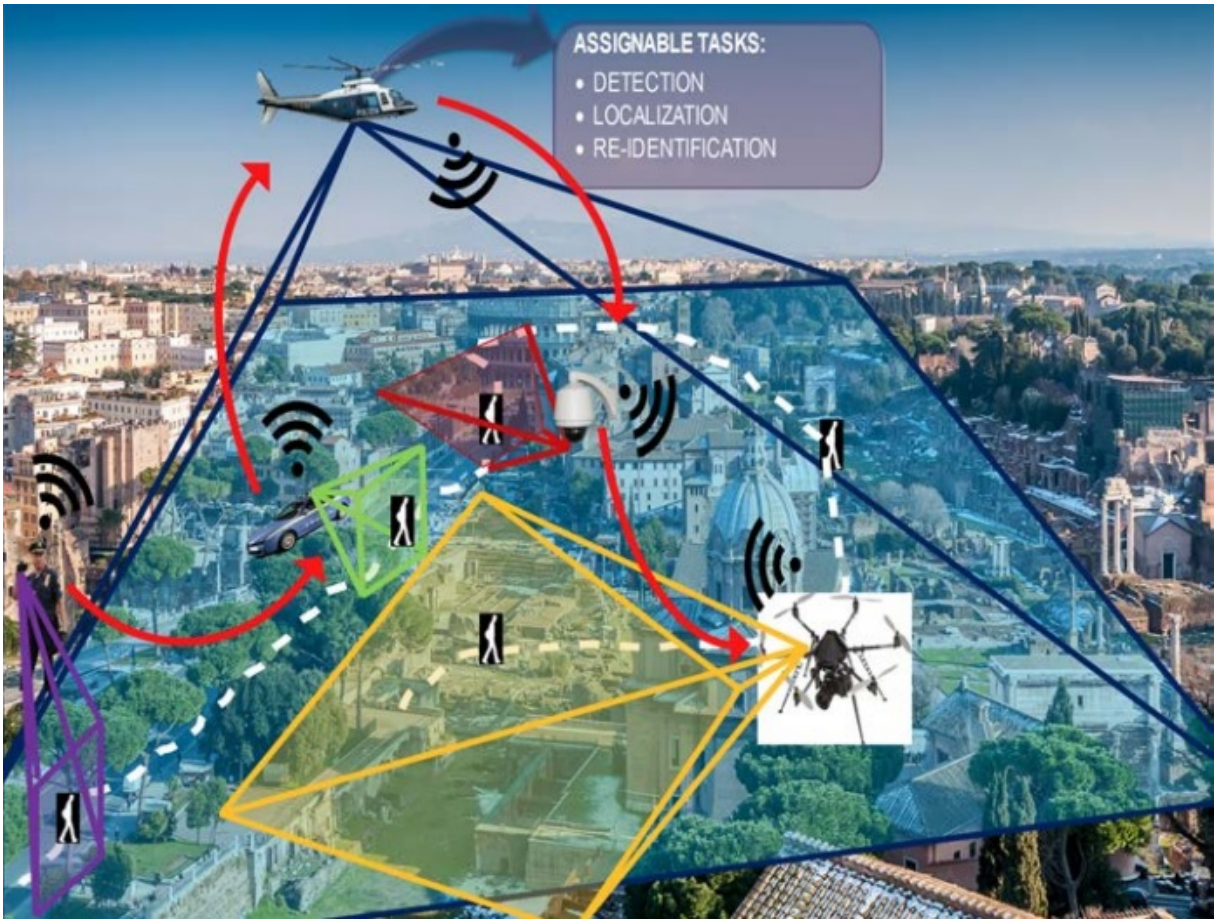


Figura 1.1: Escenario de ejemplo para una red cámara inteligente de auto-reconfigurables [3].

de configuraciones sobre las cámaras, teniendo una respuesta instantáneamente ante cualquier operación sobre ellas. Por lo tanto, el sistema deberá desarrollar un sistema de red-comunicación para cada una de las cámara por donde tramitará la información.

Las principales ventajas de realizar el estudio de los algoritmos sobre «sistemas virtuales» frente a modelo tradicional de captura de imágenes en lugares reales es, sobre todo el menor coste que conlleva, ya que únicamente se requiere de un sistema donde instalar el software y desarrollar las pruebas, otro aspecto positivo será que, si el análisis de múltiples cámaras se optase por realizar lo sobre vídeos ya generados y almacenas, esto implicaría que no se pudiese manipular las cámaras, propiedad que estos sistemas, al generarse de manera interactiva si que se lograría realizar. Por último, hay que resaltar que una re-configuración de un «sistema virtual» es mucho mas rápida y fácil de llevar acabo que en un modelo tradicional de captura de imágenes, donde múltiples aspectos han de ser considerados simultaneamente (hardware, comunicaciones,...).

1.2. Objetivo

El objetivo de este Trabajo Fin de Grado (TFG) es diseñar e implementar un entorno de pruebas interactivo para el estudio diferentes algoritmos de videovigilancia, capaces de controlar múltiples cámaras de forma remota. Para ello, el sistema deberá permitir la instalación de múltiples cámaras en espacios generados virtualmente, los cuales se desarrollaran en 3D para obtener un mayor realismo en las simulaciones.

A continuación se describen las tareas planificadas dentro del objetivo planteado:

1. Estudio del estado del arte.Análisis de herramientas para simulación multicámara y generación de entornos virtuales 3D.
2. Instalación y manejo de las herramientas selecionada, comprendiendo el funcionamiento de cada una de ellas.
3. Diseño de un sistema remoto de control de cámaras, que permita obtener datos de un simulador de entornos 3D para un posterior análisis de redes de cámaras.
4. Implementación parcial del sistema diseñado: control de cámaras sobre simulador de entornos 3D.
5. Realización de pruebas sobre el sistema implementado.

1.3. Organización de la memoria.

La estructura del documento es la siguiente:

- Capitulo 1. Motivación, objetivos y estructura de la memoria del proyecto.
- Capitulo 2. Estudio de las principales sistema multi-cámara, así como de generadores de entornos virtuales. Se realizará un estudio comparativo de cada uno de ellos explicando las principales ventajas y las conclusiones que determinan las herramientas escogidas para el TFG.
- Capitulo 3. Se describe en primer lugar el diseño preliminar llevado acabo para la realización del proyecto, después, se llevaba acabo el análisis sobre las herramientas que se van a utilizar.
- Capitulo 4. Explicación de los principales módulos realizados aportando detalles de código.
- Capitulo 5. Realización de pruebas sobre el sistema creado. Además, se realizará un estudio y conclusión de cada una de ellas tanto de los resultados como la razón por la cual se ha desarrollado

- Capitulo 6. Conclusiones extraídas de la realización del proyecto y una explicación sobre trabajo futuro.
- Bibliográfica
- Apéndices

Capítulo 2

Estado del arte

En este capítulo se presenta el estudio realizado sobre las diferentes relacionadas con el sistema propuesto en el TFG. De cada una de ella se valorará, entre otras características, que permitan el desarrollo de los objetivos establecidos en la sección 1.2.

2.1. Herramientas para simular múltiples cámaras

En esta sección se describen brevemente las diferentes tecnologías que mejor se adaptaban a la simulación de cámaras distribuídas. Estas herramientas son las siguientes: basadas en entornos virtuales (OVVV y SLNCR, sección 2.1.1) y basadas en simuladores de redes (CAMSIM, WiseMet, WSVN y M3WSN, sección 2.1.2).

2.1.1. Basados en entornos virtuales

OVVV (*ObjectVideo Virtual Tool*)[4]. Se trata de una herramienta de simulación de múltiples cámaras que permite cargar diferentes tipos de escenarios realizados en 3D con apariencia muy semejante a ambientes de la vida real. Las ventajas que proporciona son varias, entre las que se encuentra la colocación libremente de cámaras en cualquier posición de entorno virtual, y una amplia posibilidad de configurar las, además, posee herramientas de detención automática de *ground truth* , y obtiene permite generar stream en tiempo real, entre otras características. Como se muestra en la Figura 2.1, el sistema contiene dos pequeños servidores dentro de la aplicación, por un lado se encontrará el servidor de cámara cuyo objetivo será gestionar una lista de cámaras permitiendo crearlas, modificar parámetros o transmisión de frames, por otro lado se encontrará el servidor PTZ, cuya funcionalidad será controlar las siguientes cualidades de una o varias cámaras: zoom, inclinación y panorámica.

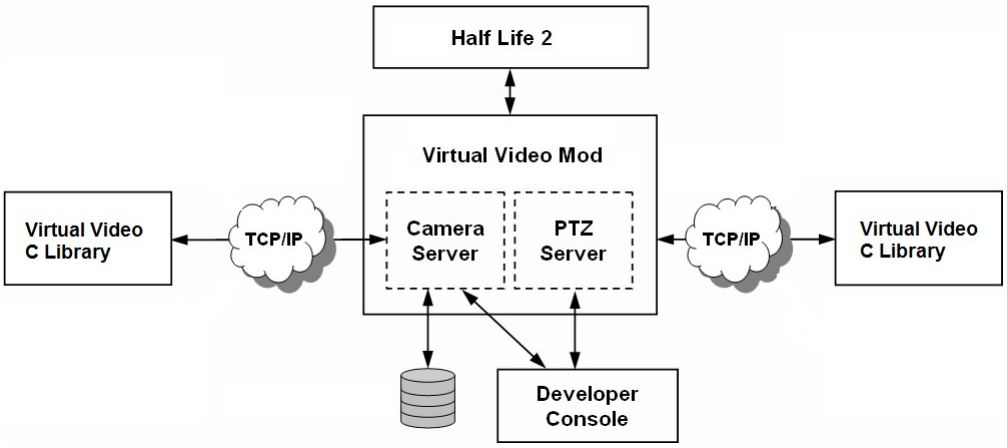


Figura 2.1: Estructura OVVV[4]

En la Figura 2.1 se muestra la estructura general del los elementos que componen al simulador OVVV. A continuación se realiza una breve explicación de los principales elementos.

- Virtual Video Mod: Extensión del videojuego HALF-LIFE 2 que añade las funcionalidades de manejo de multiples cámaras. Se encuentra formado por dos servidores: Camara Server y PTZ Server.

- Virtual Video C Library: Permite acceder a los servidores del Virtual Video Mod.

- Developer Console: Se utilizará para cargar y añadir efectos a los escenarios en 3D entre otras funcionalidades.

SLCNR *(Software Laboratory for Camera Networks Research)* **[5]**. Se encuentra formado por tres diferentes tipos de módulos, el motor del mundo virtual (VW), generador de análisis visuales (VP) y la unidad de sincronización (SYNC). VW será el respondable de la simulación de escenas en 3D , VP desarrolla la parte de detección y seguimiento sobre las animaciones en el espacio virtual y SYNC se encarga de la sinconización entre los diferentes módulos. En la Figura 2.2se muestra un resumen sobre el esquema de funcionamiento del simulador.

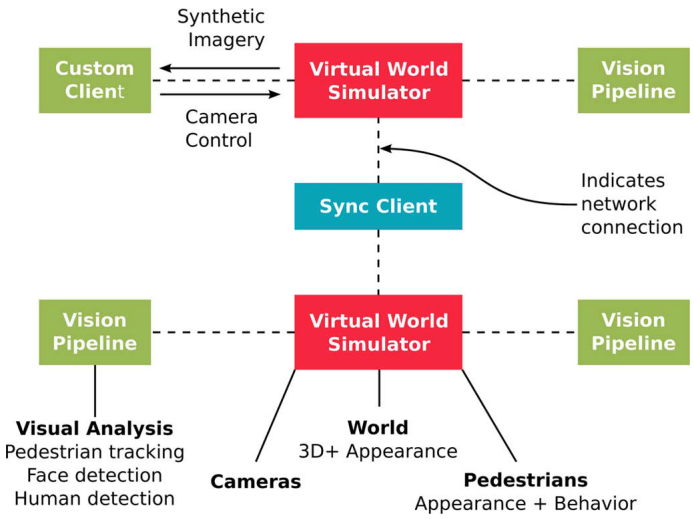


Figura 2.2: Esquema del funcionamiento del simulador virtual de cámaras (SLCNR) [5]

.

2.1.2. Basados en simuladores de redes

La principal desventaja de este tipo de simuladores es que no consideran datos visuales, es decir, no se pueden aplicar algoritmos de *Computer Vision*. Con lo cual, los fenómenos observables en las cámaras (e.g. la trayectoria de una persona que se mueve entre múltiples cámaras) se modelan mediante procesos software. A continuación se describen los simuladores estudiados.

CamSim (*Smart Camera Simulation Environment*)**[6]**. Se trata de un sistema de simulación el cual puede contener un gran número de cámaras, limitado únicamente por la memoria del sistema donde se encuentre ejecutándose. Entre las principales características del sistema se encuentra la facilidad de creación de nuevos escenarios de test, la amplitud de diferentes comportamientos que podrán tener las cámaras, así como la sencillez de implementación de nuevos algoritmos. La Figura 2.3 muestra un ejemplo de funcionamiento. Los principales elementos del sistema son los siguientes:

- Cámaras: Cada una de ellas es totalmente independiente a las demás, es decir, no existe un sistema de control de cámaras, aunque si existe un sistema de comunicación entre las diferentes cámaras. .Además, constará de un campo de visión (FOV) el cual podrá ser modificado para cada una de ellas, así como un identificador único.

- Objetos: Cada uno de ellos tendrán cualidades únicas que permitirán al sistema identificarlos . Ademas, cada objeto deberá tener una configuración inicial que determinará el movimiento en el ambiente, para ello, o bien se determina un camino completo o una dirección inicial.

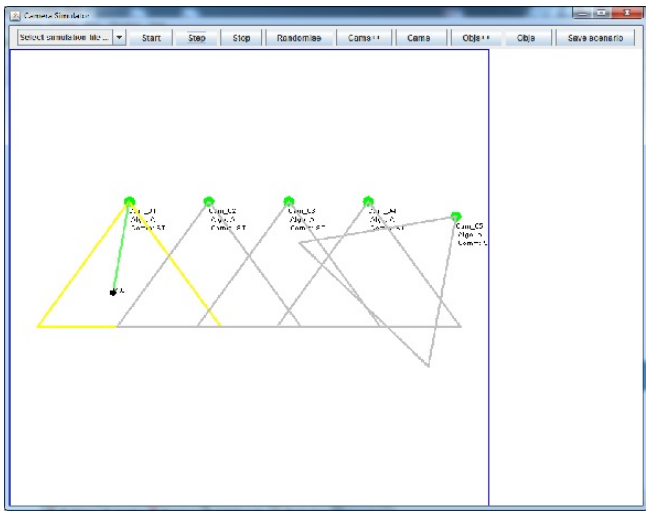


Figura 2.3: Ejemplo Pantalla CAMSIM [6]

Los siguientes tres simuladores de cámaras se encontrarán desarrollados sobre Castalia, el cual es un simulador de redes inalámbricas, y este a su vez sobre OMNET++, cuya función es la simulación de eventos.

Wise-MNet (*Wireless Simulation Environment for Multimedia Networks*)[7]. El simulador se encuentra formado por multiples nodos que representará cada uno de ellos una cámara distinta. Cada nodo (ver Figura 2.4) que se compone de cinco módulos principales, los cuales son los siguientes: *WiseBaseSensorManager*, *WiseBaseApplication*, *Communication*, *Resource* Manager y *Mobility Manager*. Los nodos a su vez se encontrarán comunicados por dos módulos: *WiseBasePhysicalProcess* encargado de capturar los datos y *Wireless Channel*/*Dummy Channel* cuya función será el envio de datos.

WSVN (*Wireless Sensors & Video Networks*)[8]. El sistema está formado por múltiples nodos pequeños, los cuales individualmente posé limitadas características teniendo poco rango de detención, espacio limitado... Sin embargo, el bajo coste individual de cada sensor facilita un mayor despliegue. Además, para aumentar su potencial, trabajan todos ellos agrupados.Las principales características del sistema son:

- Alto ancho de banda y gran de alta calidad imágenes sin pérdidas.
- Baja potencia.
- Baja latencia del tráfico de datos

En la Figura 2.5 se muestra un ejemplo de uso de la herramienta. Por un lado se encuentran los nodos o también denominados sensores que se encargan de recoger información, en otra parte se ubica el servidor que se encargará de procesar los datos recolectados, y por último, se encontrará

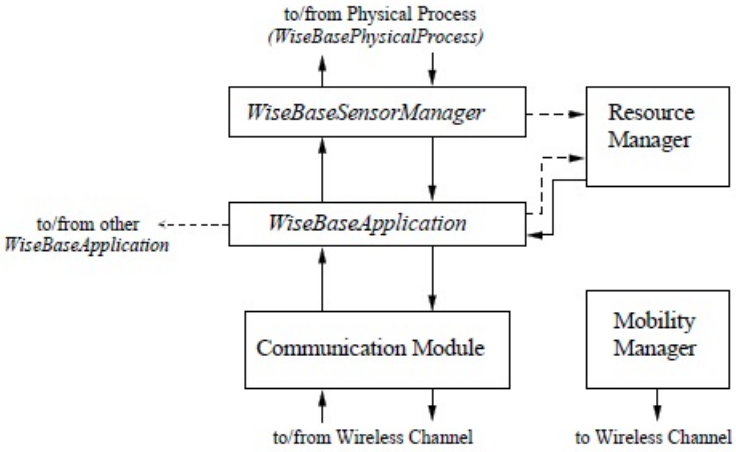


Figura 2.4: Arquitectura de los nodos del simulador WISE-MNet [7].

el sistema de captura de datos, el cual se realizará a través de Internet accediendo desde un terminal al servidor web.

M3WSN (*Mobile Multi-Media Sensor Network*)[9]. Se trata de un sistema de difusión de paquetes multimedia sobre entornos de red. Las principales contribuciones son su novedoso sistema de protocolo de comunicación, simulación sobre sistemas de vídeo transmisión, propagación multimedia de datos utilizando redes de comunicación y un sistema jerárquico multi-nivel el cual permitirá realizar detecciones de intrusiones. En la Figura 2.6 se muestra un esquema de la arquitectura:

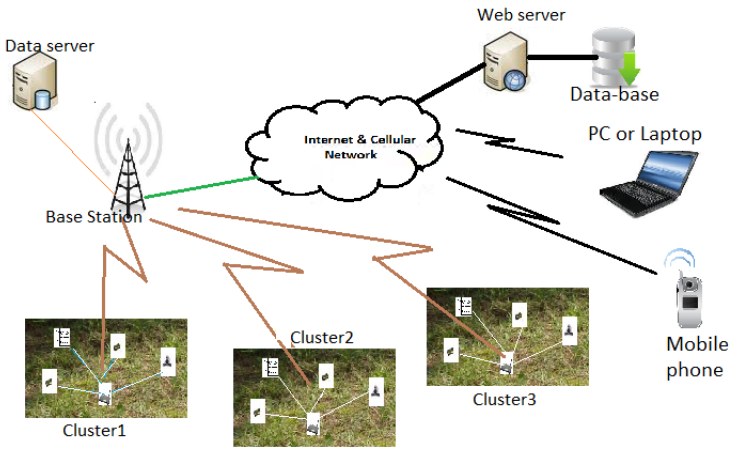


Figura 2.5: Representación de un ejemplo de uso de la herramienta WSVN[8].

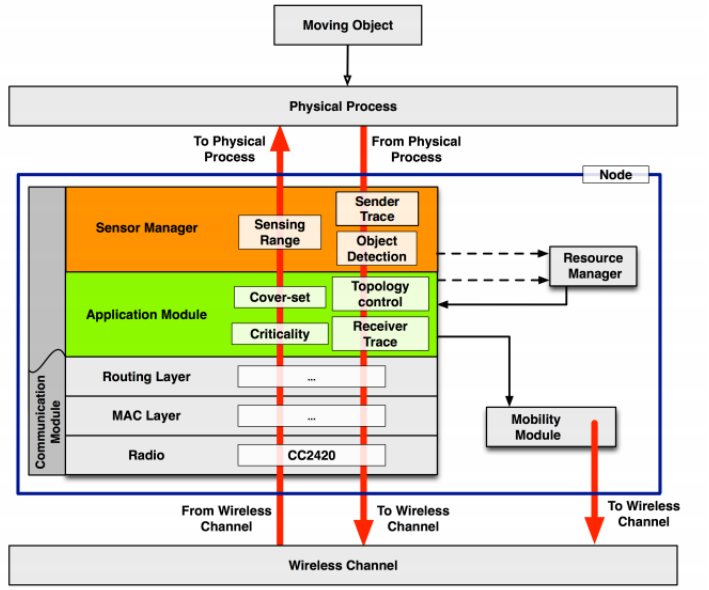


Figura 2.6: Arquitectura M3WSM[9].

2.1.3. Resumen

Las herramientas basados en entornos virtuales (OVVV y SLCNR) tienen características muy simulares, ambas se desarrollan en C++ sobre Windows, y se encuentrán enfocados para simulación de mundos virtuales 3D, la única desigualdad es que SLCNR si permite ser ampliada. De la comparativa entre las herramientas de simulación de redes (CAMSIM, WSVN, M3WSN y Wise-MNet), se concluye que al igual que las herramientas para entornos virtuales, se desarrollan en C++, sin embargo se realiza en el Sistema Operativo Linux, a excepción de CAMSIM, que utilizará el lenguaje de programación Java. Además, todos ellas simulan en 2D, con respecto a las demás carácteristicas, WSVN, M3WSN y Wise-MNet posee una gran semejanza ya que se encuentran desarrolladas sobre las mismas plataformas (OMNET++ y Castalia). En la Tabla 2.1 se muestra una comparativa de las principales características de cada herramienta de simulación.

2.2. Mundos virtuales

Una vez explicado las herramientas para el manejo de múltiples cámaras , el siguiente paso es analizar entornos que puedan cargar y editar mapas donde se instalarán las cámaras. A continuación se describen las principales herramientas analizadas.

	CÓDIGO	TIPO	RECURSOS	DESARROLLADO	COMUNICACIONES	EXTENSIBLE	ENFOCADO
OVVV	C++:Win	3D	-		-	No	Mundo virtuales
SLCNR	C++:Win	3D	-	Rutina de visión	-	Sí	Mundo virtuales
CAMSIM	Java	2D	-	-	Protocolos	Sí	Coordinación
WSVN	C++:Linux	2D	Batería, reloj, memoria	-	Wireless MAC	-	Motorización de vídeo
M3WSN	C++:Linux	2D	Batería, reloj, memoria	-	Wireless MAC	Codigo no libre	Mulimedia TX
Wise-MNet	C++:Linux	2D	Batería, reloj, memoria	Cámaras y rastreadores	Wireless, simulado	Sí	Sistema cámaras

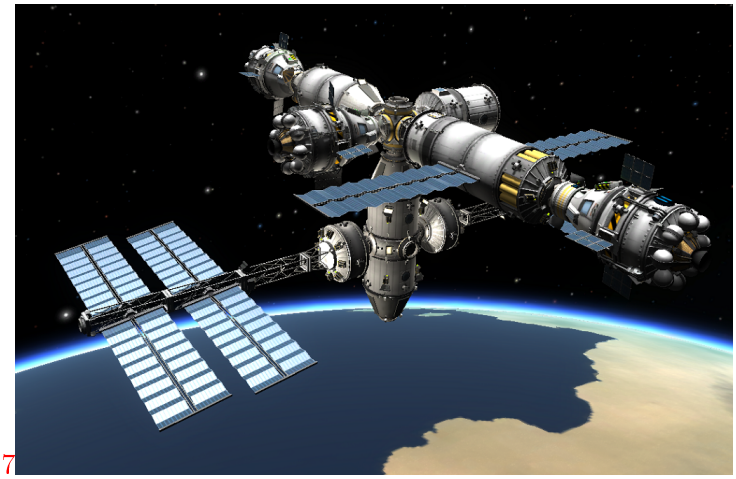
Cuadro 2.1: Herramienta de simulador de cámaras[10]

2.2.1. Tecnologías estudiadas

- **Unity**[1] : La herramienta se encuentra operativa para los sistemas operativos: Windows y OS X, y permite ejecutar los programas generados en: Windows, OS X, Linux, Xbox 360, PlayStation 3, Playstation Vita, Wii, Wii U, iPad, iPhone, Android y Windows Phone. Tiene tanto una versión gratuita de la herramienta, como una de pago, la diferencia entre ambas es que la de pago incluirá mayores opciones de manejo. El editor permite importar modelos 3D, texturas, sonido... y operar sobre ellos. Además incluye la herramienta de desarrollo *MonoDevelop* con la que podremos crear scripts en JavaScript, C#, y un dialecto de Python llamado Boo con los que se podrá extender la funcionalidad del editor.En la Figura 2.7 se muestra un ejemplo de uso.

- **Source SDK** [2] **:** Este motor gráfico permite a los usuarios compartir nuevos algoritmos para mejorar las técnicas de generación de escenarios. Incluye diversas técnicas y efectos de iluminación como renderizado, HDR, numerosos tipos de efectos ... Además, Source permite crear gran variedad de nuevos espacios virtuales, para ello utilizará el editor: Valve Hammer Editor. Esta herramienta permite diseñar una gran variedad de características básicas sobre los mapa (texturas,geometría,iluminación), colocar y editar scripts de modelos, entidades y NPCs.En la Figura 2.8 se muestra un ejemplo de uso.

[1] https://en.wikipedia.org/wiki/Unity_(game_engine)
[2] https://en.wikipedia.org/wiki/Source_(game_engine)



2.7

Figura 2.7: Escena del videojuego Kerbal Space Program desarrollado sobre Unity.

Figura 2.8: Escena del videojuego Half-life 2 desarrollado sobre Source SDK.

- **Crystal Space** [3] **:** Basado en renderizado en portales, BSP, ZBuffering y radiosividad. Entre sus principales características se destaca la gran potabilidad y su arquitectura modularizada. Además, tiene un sistema de contenedores que aporta una gran facilidad de uso, así como una abstracción de los detalles específicos, posibilitando el manejo sin necesidad de conocimiento de código. El sistema permite la instalación de plugins que hará extender las características del sistema.En la Figura 2.9 se muestra un ejemplo de uso.

- **Unreal Engine** [4] : El Motor gráfico Unreal desarrollado por la empresa Epic MegaGames se basa en una extensión del renderizado en portales conocido como Dynamic Scene Graph Technology (DSG) y BSP. Su principal característica del sistema es su arquitectura modu-

[3] https://es.wikipedia.org/wiki/CryEngine
[4] https://en.wikipedia.org/wiki/Unreal_Engine

Figura 2.9: Escena del videojuego State of Decay desarrollado sobre Crystal Space

Figura 2.10: Escena del videojuego desarrollado sobre Unreal Engine.

lar. Se encuentra disponible para las plataformas Linux, Windows, Macintosh, Playstaion 2 y Xbox. Puede utilizarse en términos de desarrollo de juegos de manera gratuita.En la Figura 2.10 se muestra un ejemplo de uso.

2.2.2. Resumen

Los sistemas de generación de espacios virtuales poseen las cualidad de desarrollarse para multitud de plataformas, y permiten utilizarse todas ellas sobre el Sistema Operativo Windows, a exceptción de Unity y Crystal Space que se podrán manejar además en otros Sistemas Operativo. Todas ellas se desarrollarán sobre C++, En la Tabla 2.2 se muestra una comparativa de los anteriores motores gráficos estudiados.

	DESARROLLADO	PROGRAMACIÓN	SISTEMAS OPERATIVOS	PLATAFORMAS
Unity	Unity Technologies	C, C++, C#	Windows, OS X	Windows, OS X, Xboz, Wii U, Play Station, iOS, Android, Blackberry...
Source	Valve Corporation	C++	Windows	Windows, OS X, Linux, Play Station, Xbox, Android...
Crystal Space	Jorrit	C++	Windows, Linux, OS X	Unix, Macintos, Windows, Rhapsody...
Unreal	Epic Games	C++, C#, Unreal Script, GLSL, Cg, HLSL	Windows	Windows, Linux, OS X, Xbox, Play Station, Wii, Android, Adobe Flash Player, HTML 5...

Cuadro 2.2: Descripción características de Motores Gráficos

2.3. Conclusiones

Para crear el simulador virtual multi-cámara en entornos virtuales se requiere selecionar varias tipos de herramientas. En primer lugar habrá que elegir un simulador multi-cámara de los descritos en la sección 2.1. El seleccionado es OVVV (sección 2.1.1), el cual además de permitir la simulación de múltiples cámaras, se diferencia de los demás por su amplio grado de configuración de cada una de ellas, incluyendo la propiedad de manejo de las cámaras durante el transcurso de la simulación. Otro aspecto importante para la elección del simulador era que permitiese la inserción de las cámaras en escenarios virtuales 3D, cosa que OVVV también lo cumple.

La siguiente herramienta que se elegirá será el motor gráfico que permita cargar, crear y editar escenarios virtuales, para ello se utilizará Source SDK (sección 2.2.1), ya que al encontrase integrado conOVVV, facilitará el diseño del sistema multi-cámara. Además, esta herramienta se adecua perfectamente al trabajo que se quiere realizar, permitiendo simular espacios virtuales con gran realismo. Otro aspecto importante que se valoró a la hora de elegir este motor gráfico es que posee un herramienta de fácil uso para la edición de mapas: Hammer.

Por último, se ha selecionado la herramienta Wise-MNet (sección 2.1.2), la cual permite integrar los puntos de vista generados en el entorno virtual sobre este simulador de redes de cámara, para realizar el análisis de diferentes algoritmos sobre los escenarios.

Capítulo 3

Análisis y diseño del sistema

3.1. Introducción

En este capítulo se detalla el análisis y diseño que se ha realizado previo a la implementación del sistema. En la sección 3.3 se explican tanto las características básicas como el diseño del sistema que se va a realizar sin entrar en detalle en la arquitectura y funcionamiento de cada una de las herramientas escogidas para realizar lo, lo cual se va a llevará acabo en la sección 4.2.

3.2. Requisitos de diseño

En este apartado se detallan los objetivos del TFG (ver sección 1.2) sobre los requisitos del sistema a desarrollar. Los requisitos se encuentran ordenados en orden progresivo con respecto a diferentes hitos a cumplir, de tal forma que para llevar acabo una nueva funcionalidad se deberán tener realizados los anteriores.

- **Hito 1**: Búsqueda y selección de herramientas.
  - Investigación de programas que permitan la simulación de múltiples cámaras en entornos de desarrollo 3D. Las cámaras deberán ser modificadas y tendrá que ser capaz de capturar imágenes.
  - Analizar herramienta que permitan tanto cargar como editar entornos virtuales en 3D obteniendo un gran realismo.
  - Elección de sistemas que mejor se adapte para el desarrollo del proyecto.
- **Hito 2**: Manejo del entorno de trabajo.
  - Instalación de las herramientas y de las librerías que se utilizarán.

- Integración entre la herramienta de simulación multi-cámara y la encargada de reproducir los escenarios virtuales.
- Familiarizarse con las herramientas. Se realizarán varias pruebas de funcionamiento las cuales cubrirán todas las siguientes opciones:
  - Creación de una cámara.
  - Modificar características que compone la cámara.
  - Movimiento de una cámara en el escenario en 3D.
  - Eliminar una cámara.
  - Captura de imágenes.
  - Realización de los cinco anteriores sobre varias cámaras.

- **Hito 3**: Desarrollo de un controlador multi-cámara.
  - Realización de un sistema remoto que permita desarrollar las operaciones descritas en el anterior apartado.
  - Adaptar el sistema creado sobre el simulador de redes de cámara: Wise-MNet (sección 2.1.2).

- **Hito 4**: Realización de pruebas sobre el sistema creado y análisis sobre cada uno de los resultados encontrados.
  - Rendimiento del simulador de redes de cámara sobre algoritmos de seguimiento y detección de personas.
  - Rendimiento de la aplicación con respecto a los siguientes términos:
    - Recursos requeridos: Memoria, CPU...
    - Conexión entre los sistemas: Ancho de banda, FPS recibidos...

3.3. Descripción general del sistema

La finalidad del proyecto será el desarrollo un sistema que permita el manejo de múltiples cámaras en un entorno de desarrollo 3D. Todo ello se realizará sobre herramientas ya creadas, y la finalidad será tanto añadir nuevas como mejorar las características que lo componen. Por lo tanto, se distinguirán al menos tres sistemas que se requieren para realizar lo:

1. Interfaz de comunicaciones: Se requiere de un sistema el cual permita la configuración e instalación múltiples cámaras.

2. Simulador de redes de cámara: La funcionalidad básica será el manejo de las cámaras.

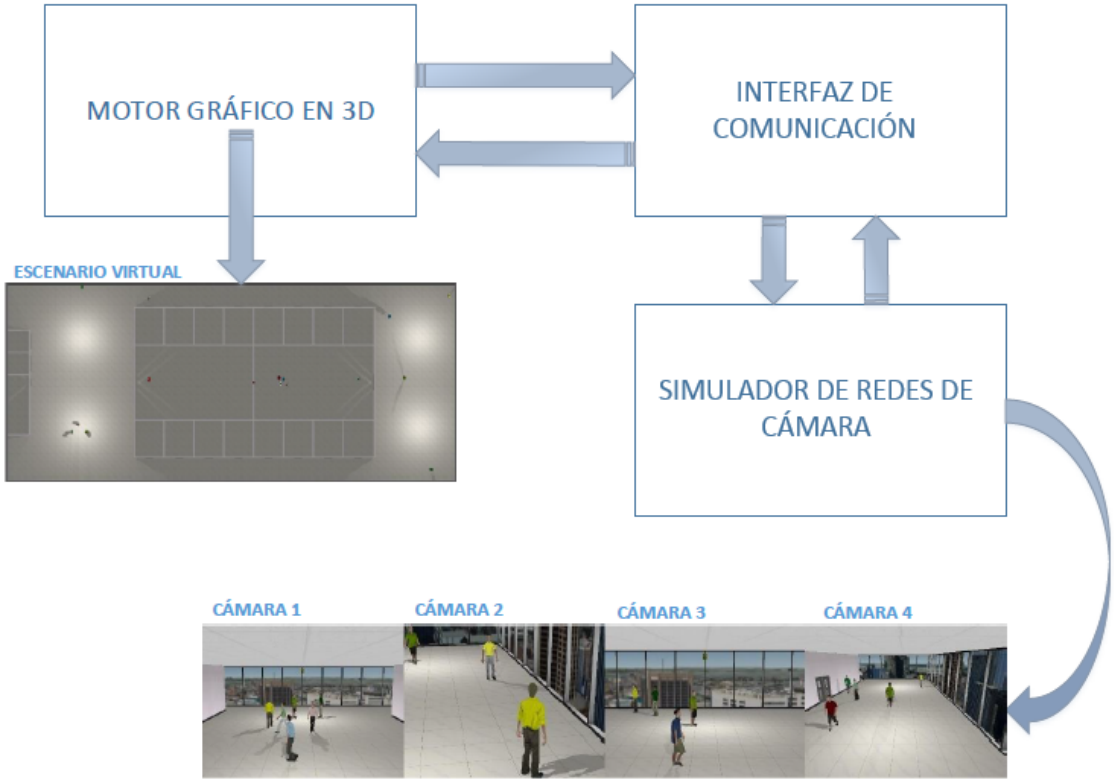


Figura 3.1: Esquema de la arquitectura general del sistema

3. Motor gráfico en 3D: Deberá permitir cargar y generar espacios virtuales cuya apariencia obtenga un gran realismo.

En la sección arquitectura 3.4 se detalla cada uno de los sub-sistemas del que se compone (Interfaz de comunicación: sección 3.4.1, Controlador: sección 3.4.2 y Motor gráfico: sección 3.4.3) . En la Figura 3.1 se muestra un esquema general de la arquitectura.

El diseño anteriormente descrito se ha estimado que supera la carga de trabajo para un Trabajo Fin de Grado , por lo que se ha optado a rebajar los requerimientos del sistema, remplazando el simulador de redes de cámara por un sistema que controle las cámaras de forma remota, al que se le asignará el nombre de: Controlador. Este sub-sistema será diseñado e implementado con el planteamiento de una futura integración al simulador de redes de cámaras, concretamente al simulador Wise-Mnet (sección 2.1.2).

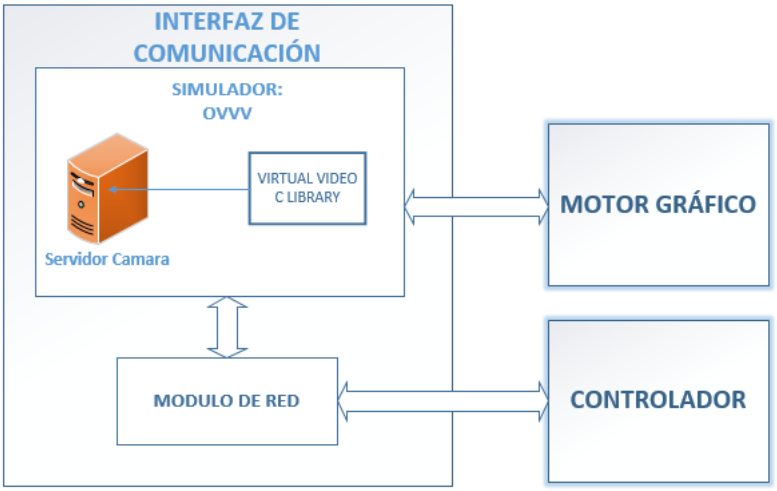


Figura 3.2: Diseño de la arquitectura del sistema: Interfaz de Comunicación

3.4. Arquitectura de los sub-sistemas

En primer lugar se detallará el modelo de cada herramientas que conformará el simulador virtual, y después se detallará el diseño para la integración. La arquitectura final se encontrará formado por los siguientes sub-sistemas: Interfaz de comunicación (sección 3.4.1), Controlador (sección 3.4.2) y Motor Gráfico 3D (sección 3.4.3).

3.4.1. Interfaz de comunicaciones

3.4.1.1. Módulos

El sistema de Interfaz de comunicaciones se encuentra formado por el simulador multi-cámaras OVVV más un modulo de red. El objetivo del **módulo de red** es la creación de un canal entre el Controlador y el sistema de Interfaz de Comunicación por el cual se envíe solicitudes del Controlador al Motor gráfico, y las repuestas del Motor Gráfico al Controlador. El **simulador multi-cámara OVVV** tiene como funcionalidad controlar el manejo de las cámaras instaladas en el Motor Gráfico.En la Figura 3.2 se muestra un esquema de la arquitectura.

3.4.1.2. Funcionalidad

La Interfaz de comunicaciones tiene dos funcionalidades principales, la primera de ellas es comunicar el Controlador con el Motor Gráfico, y la segunda funcionalidad es tener el control sobre las cámaras.

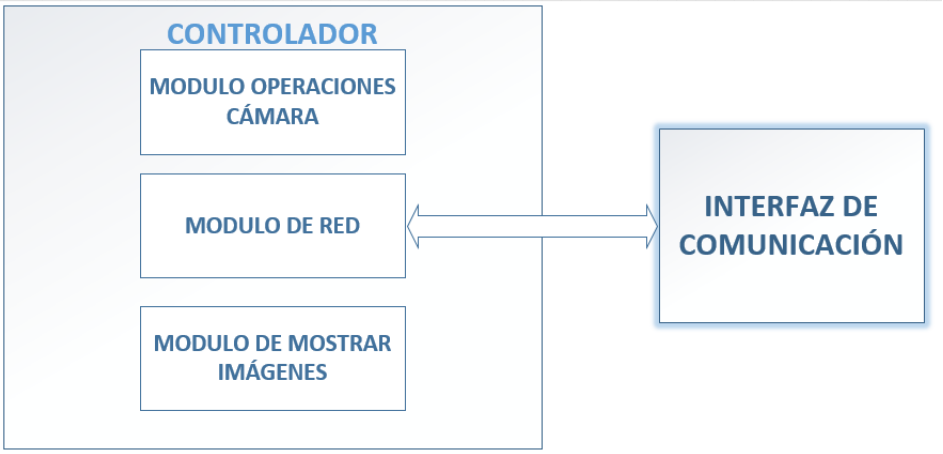


Figura 3.3: Diseño de la arquitectura: Controlador.

3.4.2. Controlador

El sistema Controlador es el encargado del manejo de las cámaras.En la sección 3.4.2.1 se explica los módulos de los que estará formado y en el la sección 3.4.2.2 explica las funcionalidades que permite realizar en cuanto al control de las cámaras.

3.4.2.1. Módulos

El **módulo operaciones de cámara** será el encargado del manejo de las cámaras, el **módulo de red** se encarga de enviar las peticiones requeridas de las cámaras, y por último, el **módulo de mostrar imágenes** tiene como objetivo la representación de los frames recibidos de cada cámara. En la Figura 3.3 se muestra un esquema de esta arquitectura descrita.

3.4.2.2. Funcionalidad

En la Figura 3.4 se muestras las funcionalidades que permitirá realizar el Controlador con respecto al manejo de las cámaras. A continuación se explicará cada una de ellas:

1. Menú: Punto de partida del programa que comenzará una vez se haya establecido la conexión entre el Controlador y el Simular multi-cámara. En este punto se escogerá una de las tres posibilidades que presenta el programa: Crear Cámara2, Listar Cámara 3y Salir4.
2. Crear Cámara: Permite crear una cámara en cualquier posición del mundo virtual.
3. Listar Cámara : Aparece una lista de las cámara existentes
4. Salir: Cierra la conexión entre el Controlador y el Simulador multi-cámara.
5. Capturar todas cámaras: Muestra todas las imágenes capturados por cada cámara.

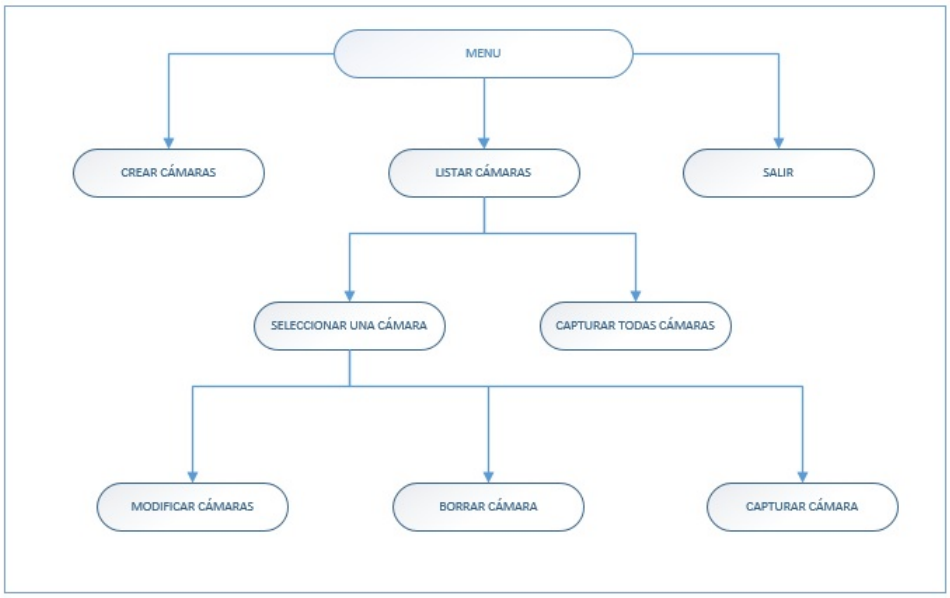


Figura 3.4: Diagrama de funcionamiento Controlador

6. Seleccionar una cámara: Se deberá escoger entre una de las siguientes operaciones que se quiera realizar sobre la cámara: Modificar cámara7, Borrar cámara 8 o Capturar cámara 9.

7. Modificar cámara: Modificará uno o varios parámetros de la cámara.

8. Borrar cámara: Eliminará la cámara.

9. Capturar cámara: Se mostrará frames de la cámara seleccionada, además, se podrá mover la cámara por el escenario en 3D al mismo tiempo.

3.4.3. Motor Gráfico en 3D

3.4.3.1. Módulos

La herramienta Motor gráfico se encuentra formado por los módulos de red y simulación de mapas. El **modulo de red** tiene como objetivo crear un canal de comunicación entre el Motor gráfico y el Controlador pasando por el sistema de Interfaz de comunicación. El **módulo de simulación de mapas 3D** permite en primer lugar cargar los escenarios virtuales, y después, de la instalación de cámaras en el espacio. En la Figura 3.5 se muestra un esquema de la arquitectura del Motor gráfico.

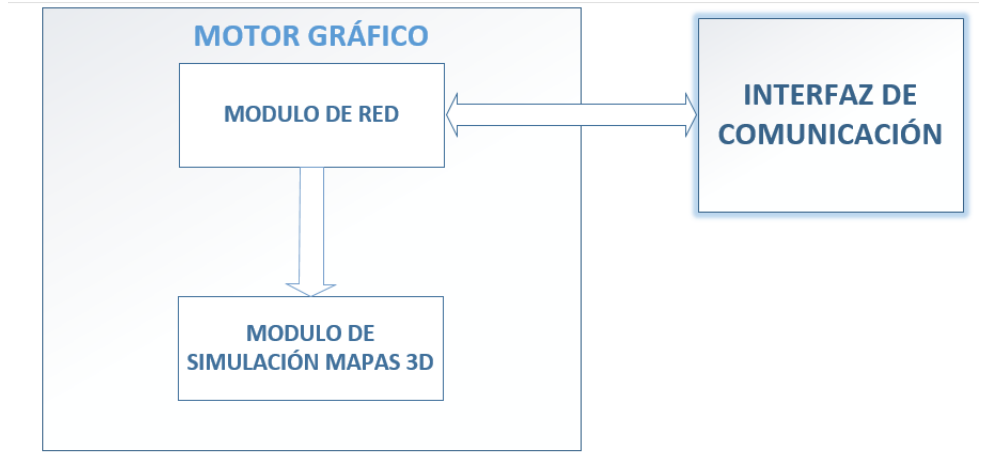


Figura 3.5: Diseño de la arquitectura: Motor Gráfico 3D.

3.4.3.2. Funcionalidad

Las funcionalidades del Motor Gráfico quedan dividido en dos partes:

- Mantener la simulación del escenario virtual.
- El desarrollo de la comunicación con el Interfaz de Comunicación, que tendrá como objetivo devolver la información solicite la Interfaz.

3.5. Integración: Simulador Virtual para sistemas multi-cámara distribuidas.

El estructura final de la arquitectura quedará formador por la agrupación de los siguientes sistemas: Interfaz de comunicación, Controlador y Motor gráfico. La funcionalidades de cada uno de ellos serán:

- Controlador: Manejo de las cámaras y representación de imágenes.
- Motor gráfico: Simulación de instalación de cámaras en espacios virtuales.
- Interfaz de comunicación: Realización de un canal de intercambio de mensajes entre el Controlador y el Motor gráfico.

3.5.1. Interfaces.

El Simulador Virtual diseñado consta de tres sub-sistemas, lo que aporta una gran relevancia el sistema de comunicación entre los mismos. Para llevarlo acabo se realiza para cada sub-sistema

Figura 3.6: Diseño de la arquitectura: Simulador Virtual para sistemas multi-cámara distribuidas.

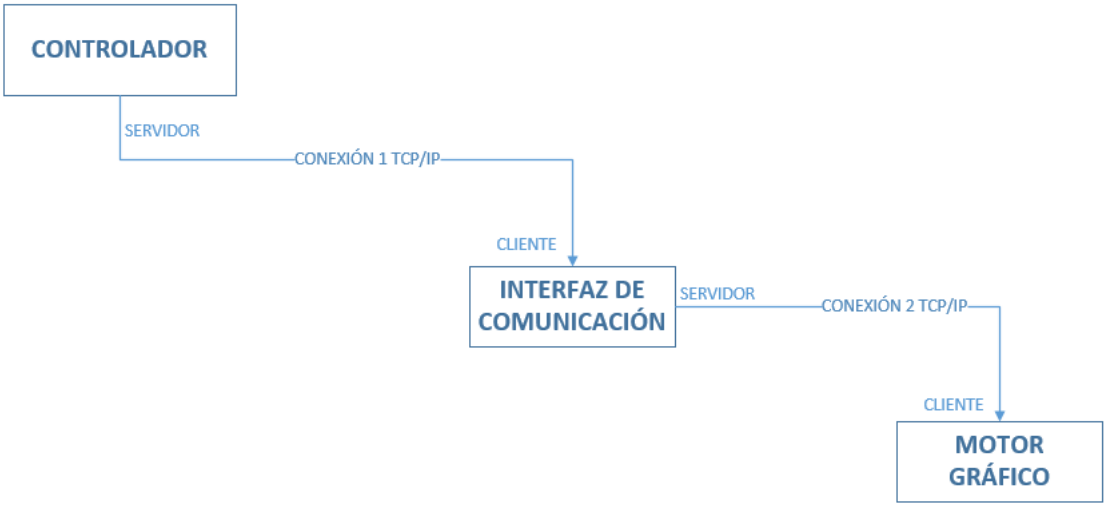


Figura 3.7: Interfaz de comunicación entre los sub-sistemas

un módulo de red, cuya funcionalidad será el envío de información para cada uno de ellos. En total habrá dos conexiones existentes,mientras los sub-sistemas Controlador y Motor gráfico tendrán una interfaz, la Interfaz de Comunicación tendrá dos, que los unirá con los anteriores. La relación entre los sub-sistemas será de cliente-servidor para el tipo de conexión TCP/IP, de tal modo que entre el Controlador y la Interfaz de Comunicación el servidor sea el Controlador y la Interfaz de Comunicación sea el cliente, y para la conexión Internet de Comunicación con el Motor gráfico, el servidor será la Interfaz de Comunicación siendo el Motor gráfico el cliente. En la Figura 3.7 se una imagen de como queda formado la conexión explicada.

3.6. Diagrama funcionamiento.

En la siguiente Figura 3.8se muestra cada iterración que tendrán entre los diferentes herramientas durante el transcurso de su utilización. Contará de tres fases: Creación de conexión e inicialización, desarrollo de operaciones sobre las cámaras y terminación de conexión. En el periodo de creación de conexión e inicialización se establecerá en primer lugar el sistema de comunicación por los cuales las herramientas se enviarán información, una vez finalizado esta etapa, cada sistema realizará una inicialización del sistema, en el caso de simulador de redes y el simulador multi-cámara cada uno se encargará de inicializar las estructuras que se vayan a utilizar, mientras que en el motor gráfico 3D, la herramienta cargará el escenario sobre el cual se van a desarrollar las pruebas. Antes de terminar esta fase, el simulador de redes cargará un fichero inicializador de cámaras, el cual será enviado en primer lugar al simulador multi-cámara, y una vez haya actualizado los valores que correspondan, se pasará la información al motor gráfico para que opere sobre el escenario.

La segunda etapa, llamada desarrollo de operaciones sobre las cámaras, tendrá como funcionalidad la manipulación de las cámaras instaladas en el escenario virtual según las ordenes marcadas por el simulador de redes. Esta fase tendrá lugar hasta que que uno de los sistemas determine que quiere cerrar la vinculo con los demás componentes y pasará a la última fase: terminación de conexión.

La última etapa tendrá como objetivo cerrar los canales de comunicación que se hayan creado durante la primera fase.

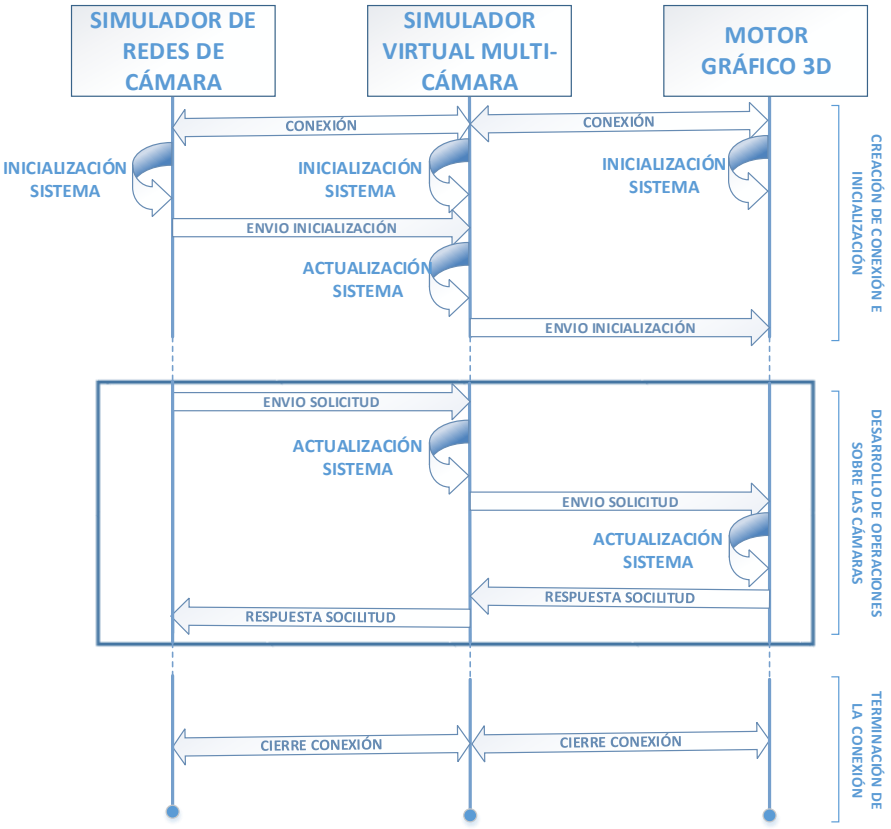


Figura 3.8: Diseño de la arquitectura del simulador de redes de cámara

Capítulo 4

Implementación del sistema

4.1. Introducción

En este capítulo se explica el desarrollo de programación llevado acabo sobre los principales módulos implementados. Los entornos de programación elegidos son Visual Studio 2010 (Windows) y editor de texto combinado con el compilador GCC (Ubuntu) para la parte de desarrollo del controlador.

4.2. Simulación remota con múltiples cámaras utilizando OVVV sobre entornos 3D

Mientras que en el capítulo anterior (ver sección 3.3) se explico el diseño general que tiene la aplicación, en este apartado donde se explica en detalle sobre las herramientas seleccionas. El sistema esta formado de la siguiente manera, la Interfaz de Comunicación se desarrollará sobre OVVV, el Motor Gráfico es el Source SDK y el Controlador será un sistema creado para una futura integración con el simulador de cámaras Wise.

4.2.1. Interfaz de Comunicación

Como ya se comentó (sección 2.3), la herramienta seleccionada para la Interfaz de Comunicación será: OVVV. Esta herramienta se compone de dos elementos:

- Vídeo Virtual Mod: Se trata de una extensión del videojuego HALF-LIFE 2, la cual incorpora la posibilidad de configurar y extraer imágenes de múltiples cámaras en entornos virtuales.
- Vídeo Virtual API: Esta librería permite el manejo con las cámaras instaladas en el Video Virtual Mod. En el Anexo B.1 se explica con mayor detalle.

Arquitectura Como se muestra en la Figura 4.1 , el simulador OVVV se trata de una extensión del videojuego Half-Life 2. El simulador tendrá dos pequeños servidores, Servidor de Cámara y Servidor PTZ, pero para el desarrollo de nuestro sistema de simulación únicamente se utilizará el servidor:Servidor de Cámara, el cual Se encarga de mantener una lista de cámara instalada en el escenario. Los clientes que se conecten al servidor podrán: crear nuevas cámaras, modificar parámetros, desconectar cámara, cambiar posición... entre otras funcionalidades.

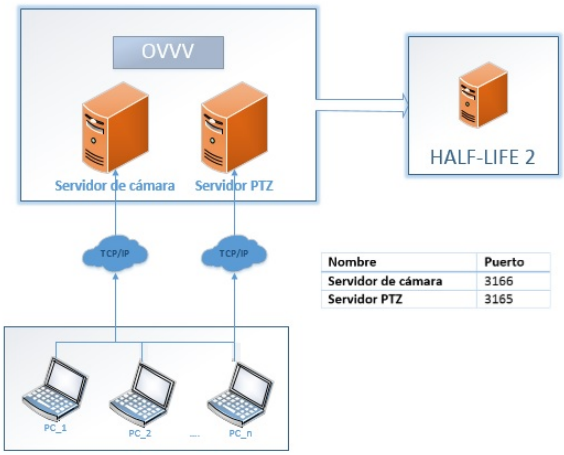


Figura 4.1: Arquitectura OVVV

4.3. Motor Gráfico 3D: Source

La herramienta Source permite simular espacios virtual generados por su editor de mapas: Hammer. A continuación se realizará una explicación de este manipulador de mapas en 3D.

Hammer: contiene dos elementos principales: brushes y entities. Las primeras harán referencia a la geometría del mapa incluyendo paredes, pisos, cielo, etc., así como la geometría no visible (cortes del terreno) , por otro lado, se encuentran las entities que sirven una amplia gama de propósitos, tales como la definición de la geometría estática y dinámica que incluye muebles, árboles, gente y otros elementos, además del control de las fuentes de luz, sombras y reflejos. El proceso de generación de un entorno sera el siguiente, en primer lugar se generará el fichero: .vmf, y una vez creado los brushes y entities, se deberá compilar, y si todo se ha realizado correctamente, se generara el fichero: .bsp, el cual ya podrá ser utilizado para cargarlo en el simulador. En la Figura 4.2 se muestra la interfaz de esta herramienta.

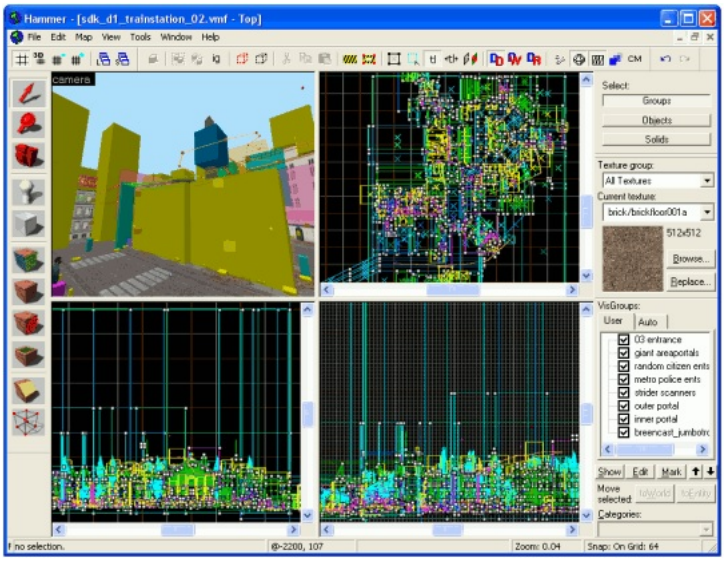


Figura 4.2: Interfaz editor Hammer.[1]

4.3.1. Controlador

Uno de los objetivos (sección 1.2) finales del Trabajo Fin de Grado es adaptar el simulador OVVV al simulador de redes Wise. Para ello, primero crearemos un paso previo creando un programa nombrado como: Controlador. El cuál se encontrará operando sobre el Sistema Operativo Linux, para así reproducir las mismas condiciones donde se ejecuta Wise.

4.3.2. Integración

La última parte queda explicar como el Controlador (sección 4.3.1) se integra con el Simulador en el Entorno Virtual (sección **??**). Para realizar lo se tendrá que crear una nueva conexión entre el Controlador y Simulador. Además, en la parte del Simulador llamada : Aplicación Manejo de Cámara, se añadirá una nuevo módulo, que se encargará tanto de controlar la conexión entre el Controlador y el Simulador, como de trasmitir las ordenes (ver Figura 3.4) requeridas entre el Controlador al Simulador OVVV.

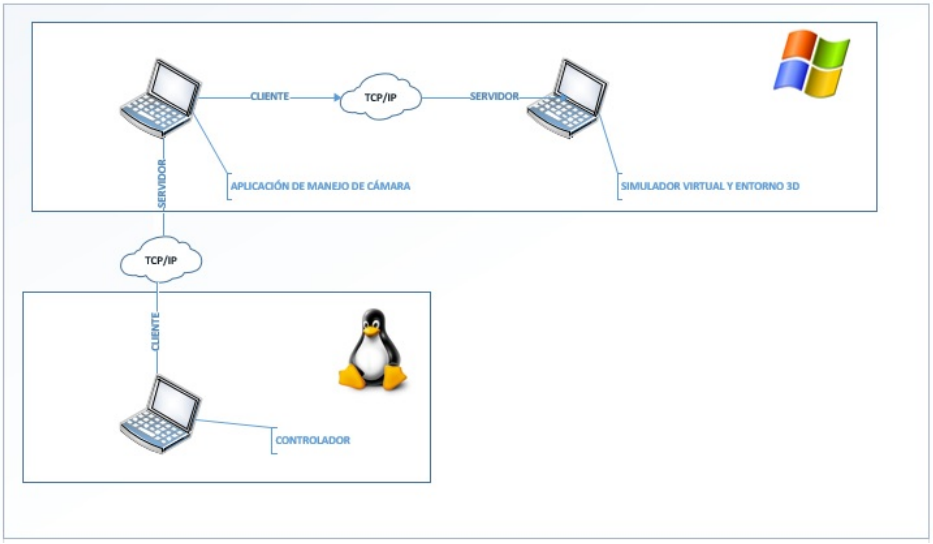


Figura 4.3: Estructura Controlador-Simulador.

4.4. Conexiones

En la Figura inferior 4.4se muestra las dos conexiones que existirán. Mientras que la del controlador multi-cámara con la aplicación de manejo de multi-cámara se realiza entre distintos ordenadores, la otra conexión realizada entre la aplicación de manejo de multi-cámara con el espacio virtual se realiza sobre el mismo terminal.

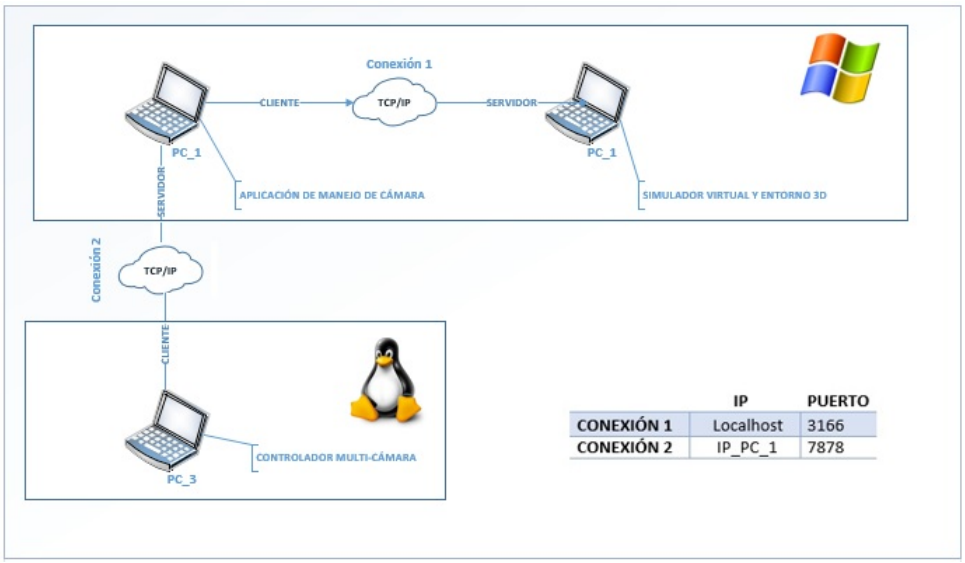


Figura 4.4: Conexión ordenadores

Las conexiones se tratan de tipo TCP/IP y viene descritas su comportamiento por la Figu-

ra4.5. Las funciones en cada paso serán las siguientes:

1. Comunes entre cliente-servidor:

   a) Crear socket:

      1) Descripción: La función socket crea un extremo de una comunicación y devuelve un descriptor.
      2) Código ejemplo: $idSocked = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP)$;

   b) Enviar:

      1) Descripción: La función *send* escribe hasta *longitud* de *buff* al socket especificado por *idSocked*.
      2) Código ejemplo: *send(idSocked, buff, longitud, 0);*

   c) Recibir:

      1) Descripción: La función *recv* lee datos del socket especificado por *idSocked*.
      2) Código ejemplo: *recv(idSocked, buff, longitud, 0);*

   d) Cerrar conexión:

      1) Descripción: La función close cierra un socket. Su argumento es el identificador del socket.
      2) Código ejemplo: *close(idSocked);*

2. Solo para el cliente:

   a) Conectar:

      1) Descripción: La función *connect* solicita poder conectar el socket especificado por *idSocket* a un socket remoto que es especifico en *serv_addr*.
      2) Código ejemplo: *connect (idSocket, serv_addr, addrlen);*

3. Solo para el servidor:

   a) Construir:

      1) Descripción: La función *bind* asocia el socket dado por *sockfd* a la dirección local especificada por *addr* para que el socket quede asignado al puerto especificado en la misma.
      2) Código ejemplo: *bind(idSocked, sockfd, addr);*

   b) Escuchar:

1) Descripción: La función listen especifica que el socket dado por *idSocked* desea aceptar conexiones.

2) Código ejemplo: *listen(idSocked, 3);*

c) Aceptar:

1) Descripción: La función *accept* acepta una petición de conexión al socket especificado por *idSocked.*

2) Código ejemplo: *accept(idSocked, addr, addrlen);*

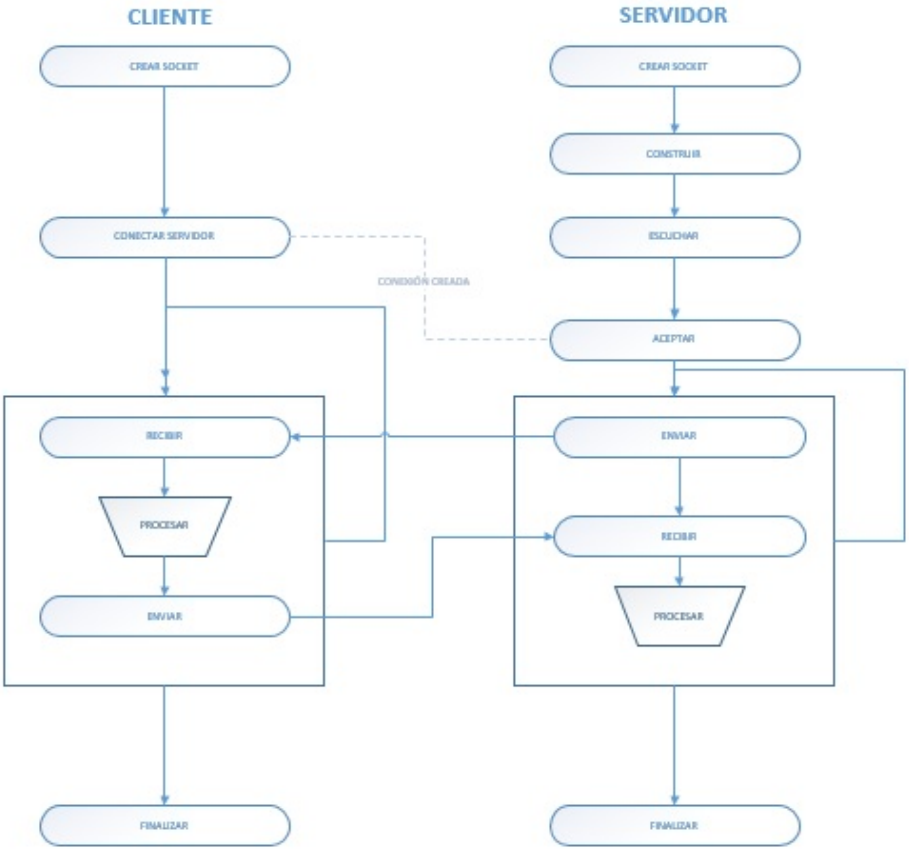


Figura 4.5: Conexión cliente-servidor TCP.

4.5. Manejo de las cámaras

En está sección se detalla el pseudocódigo implementado de las principales funciones realizadas por el Sistema editor de cámaras y por el Controlador.

```
typedef struct vv_Camara_t{

VVCamera camera[MAX_CAM]; /* Estructura que contiene un total de MAX_CAM de cámaras */
int numCamera;            /* Contiene el número total de cámaras conectadas */
int identificador;        /* Almacena el proximo identificador*/

}CAM;
```

Figura 4.6: Pseudocódigo estructura: CAM.

4.5.1. Sistema editor de cámaras

De este sistema se explica el pseudocódigo de la estructura agrupa todas las cámaras 4.5.1.1, y las funciones de inicialización del sistema 4.5.1.2, creación de una nueva cámara 4.5.1.3 y envío de una imagen 4.5.1.4.

4.5.1.1. Estructura cámara:

La estructura CAM almacenará un conjunto de cámaras definida a su vez por la estructura VVCamera que se encuentra desarrollada en la librería Virtual Video C B.1, además contendrá un campo que contiene el numero total de cámaras y otro campo que almacena el siguiente identificador que se asignará en la creación de una nueva cámara. En la Figura 4.6 se muestra el pseudocódigo de la estructura.

4.5.1.2. Función inicializadora del sistema:

Esta función que se llama; InicializaciónSistema, tiene como primer objetivo establecer la conexión con el mundo virtual y después inicializar la estructura de la cámara 4.5.1.1 por los parámetros establecidos en el fichero inicializar 4.5.2.2 que se encuentra en el sistema controlador. En la figura 4.7 se muestra el pseudocódigo de la función.

4.5.1.3. Función creación de una cámara:

Esta función será llamada cada vez que se quiera crear una nueva cámara. Como se muestra en la Figura 4.8, los primeros pasos que se realizarán serán: conectar la cámara, asignar un identificador y el tipo de la cámara, después, recibe los demás parámetros que conformarán la cámara, los cuales serán insertados en la cámara por medio de la función: cambiarRestoValores (ver Figura 4.9). Por último, se actualizarán los valores de la estructura CAM 4.5.1.1.

```
InicializacionSistema (EstructuraCamara CAM, int idSocket){

        /* Función necesario para la inicialización del mapa*/
        VVInitialize()

       /* Recibir fichero del controlador para inicializar cámaras */
        recibirMensaje(mensaje, idSocket)

        /* Inicilizar estructura*/
        Inicializarcamaras(CAM, mensaje)

        /* Salir */
        Return
}
```

Figura 4.7: Pseudocódigo función: InicializacionSistema.

```
CrearCamara (EstructuraCamara CAM, int idSocket){

      /* Conectar cámara */
      CamConnect(CAM.camera[CAM.numCamera], "localhost")

       /* Asignar id cámara */
      CamSetID(CAM.camera[CAM.numCamera],CAM.identificador)

       /* Asignar tipo cámara cámara */
      CamSetType(&V_cam->camera[V_cam->numCamera], VV_STDCAM, VV_TRUE, 0, NULL)

      /* Recibir resto de parámetros */
      recibirMensaje(mensaje, idSocket)

      /* Modificar resto parámetros */
      cambiarRestoValores(CAM, mensaje)

      /* Actualizar estrucuta CAM */
      CAM.numCamera++;
      CAMidentificador= proximoIdentificador(CAM)

       /* Salir */
       Return

}
```

Figura 4.8: Pseudocódigo función: CrearCamara.

```
cambiarRestoValores (EstructuraCamara CAM, cadena mensaje){

        /* Obtener valores nuevos */
        valores = obtenerValores(mensaje)

        /* Cambio valores intrinsecos */
        CamSetIntrinsicParameters(CAM.camera[CAM.numCamera], valores.ancho, valores.alto,
valores.framerate, valores.FOV)

        /* Cambio valores extrinsecos */
        CamSetExtrinsicParameters(CAM.camera[CAM.numCamera], valores.X, valores.Y,
valores.Z, valores.RX, valores.RY, valores.RZ)

        /* Salir */
        Return

}
```

Figura 4.9: Pseudocódigo función: cambiarRestoValores.

4.5.1.4. Función envío de una imagen:

La función: envíoImagen, tiene como objetivo la captura y envío de una imagen al controlador. Para ello se seguirán los pasos detallados en la Figura 4.10. Lo primero que se realiza es el envío al controlador del tamaño de la imagen a enviar, después, se esperara recibir la aceptación del envío de las imágenes. Una vez admitido el controlador recibir las imágenes, el sistema entrará en un bucle que no terminará hasta que el controlador no lo determine. Dentro del bucle los pasos serán, primero capturar la imagen y prepararla para ser enviada, después, el sistema quedará a la espera de una respuesta, que podrá ser de modificación de parámetros de una cámara, requerimiento de una nueva imagen o de fin del envío de imágenes.

4.5.2. Controlador

Del Controlador se especificará en primer lugar la función encargada de recibir y mostrar las imágenes de las cámaras 4.5.2.1, y en segundo lugar se explica el fichero que inicializa las cámaras 4.5.2.2.

4.5.2.1. Función recibir y mostrar imagen:

La función se llamara; recibirImagenMostrarCámara, y tendrá como único paramento el identificador del socket. La primera función que realiza al comenzar es el tamaño de la imagen que va a tomar, este valor será útil a la hora de crear la ventana donde se insertará la imagen, lo

```
envioImagen (EstructuraCamara CAM, int idSocket){

        /* Envío tamaño de la imagen */
        TamImagen = CamGetIntrinsicParameters(CAM.imagez)
        envioMensaje(TamImagen, idSocket)

        /* Recibo: OK del controlador */
        recibirMensaje(mensaje, idSocket)

        /* Bucle envío imagen */
        mientras ( noFin ){

                /* Obtener imagen */
                CamGetLatestFrame(CAM.camera[CAM.numCamera].img, VV_TRUE)

                /* Preparar envío imagen y  la envío */
                imagen = prepararImagen(CAM.camera[CAM.numCamera].img)
                envioMensaje( imagen, TamImagen, idSocket);

                /* Recibo mensaje con las siguientes posibles opciones:
                        - Nueva imagen.
                        - Modificar cámara.
                        - Fin
                */
                reciboMensaje(mensaje, idSocket );


                /* Modificar la cámara y volver a obtener a ir a obtener nueva imagen*/
                si mensaje = modificar cámara
                        modificarCamara(CAM)

                /* Volver a ir obtener nueva imagen*/
                si mensaje = NuevaImagen

                /* Salir */
                si mensaje = Fin
                        Return
}
```

Figura 4.10: Pseudocódigo función: envioImagen.

```
recibirMostrarImagenCamara (int idSocket) {

        /* Obtener tamaño de la imagen*/
        reciboMensaje(TamImagen, idSocket);

        /* Bucle para recibir mostrar imagen*/
        mientas ( noFin ){

            /* Recibir imagen*/
            reciboMensaje(Imagen, idSocket);

            /* La primera iteración crea ventana donde insertar imagen*/
            si Iterraccion = 0
                crearVentana(contenedor, TamImagen) /* Crear ventana*/
                insertarImagen(contenedor,Imagen)  /* Insertar imagen*/
                mostrarVentana(contendor)          /* Mostrar imagen*/
                Iterraccion = 1
            Sino
                insertarImagen(contenedor,Imagen)
                mostrarVentana(contendor)

            /* Comprobar interrupcion*/
            Si interrupcion i= 0  /* En caso de que haya interrupcion*/
                /* Comprobar tipo interrupcion*/
                Si interrupcion = Fin
                    /* Solicitar fin de recibo de imagenes */
                    envioMensaje(FIN, idSocket);

                Si interrupcion = Mover
                    /* Solicitar mover imagen */
                    envioMensaje(Mover, idSocket);

            sino /* En caso de que no haya interrupcion*/
                /* Solicitar nueva imagen */
                envioMensaje(NuevaImagen, idSocket);

}
```

Figura 4.11: Pseudocódigo función: recibirImagenMostrarCámara.

cual se explicará mas adelante. Después entra en un bucle, que se mantendrá en el hasta que el propio Controlador no genere una interrupción de $<<Fin>>$. Dentro del bucle lo primero que realiza es recibir la imagen, para después insertarla sobre la ventana, una vez mostrada, sino se ha recibo ninguna interrupción volverá al principio del bucle, en caso de recibir una interrupción $<<Modificar\ cámara>>$, envía la solicitud al sistema editor de cámaras para que realice las modificaciones sobre la cámara y vuelve a esperar a recibir una nueva imagen. Como se observa, esta función es complementaria a la explicada en el punto 4.5.1.4.En la Figura4.11 se muestra el pseudocódigo de la función.

Camaras.ini

```
1 // Fichero que inicializa las cámaras
2
3 NUM_CAMARAS=2
4
5 PARAMETROS_CAMARA=1
6 ANCHO=200
7 ALTO=400
8 FOV=48
9 FPS=10
10 X=10
11 Y=25
12 Z=4
13 RX=6
14 RY=7
15 RZ=8
16 FIN_PARAMETROS_CAMARA
17
18 PARAMETROS_CAMARA=2
19 ANCHO=100
20 ALTO=100
21 FOV=48
22 FPS=10
23 X=100
24 Y=5
25 Z=46
26 RX=62
27 RY=70
28 RZ=0
29 FIN_PARAMETROS_CAMARA
30
```

Figura 4.12: Pseudocódigo fichero: Camaras.ini.

4.5.2.2. Fichero inicializador de cámaras:

El fichero inicializador de cámaras creado con el nombre de; Camaras.ini, tiene la funcionalidad de que, una vez establecido la conexión entre todas las herramientas, permita crear las cámaras definido en el fichero. En la Figura4.12 se muestra un ejemplo de este fichero. Las líneas que comiencen por // serán interpretadas como comentarios, la variable NUM_CAMARAS, vendrá precedida del número total de cámara a inicializar, FIN_PARAMETROS_CAMARA definirá la terminación de los parámetros de una cámara y los demás valores corresponden a los diferentes parámetros que conformarán una cámara.

Capítulo 5

Experimentos

5.1. Introducción

En este capítulo, primero se presentan las herramientas y los sistemas que se han utilizado para la realización del TFG (sección 5.2), y después se realiza un análisis de rendimiento de la aplicación diseñada. Se han realizado cuatro experimentos detallados en las secciones 5.3 hasta 5.6.

5.2. Herramientas utilizadas en los experimentos

En la siguiente tabla 5.1 se muestra los programas utilizados para la realización del trabajo fin de grado.

A continuación se describe las características de los ordenadores utilizados para la realización del Trabajo Fin de Grado.

- **Simulador virtual:** Sistema Operativo: Microsoft Windows XP Profesional Versión 2002. Procesador: Intel(R) Pentium(R) D CPU 2.80GHZ. Memoria RAM: 1 GB

- **Controlador:** Sistema Operativo: Ubuntu 12.04.05 LTS. Procesador: 2xIntel(R) Core (TM) 2 Duo CPU E7500 2.93GHz. Memoria RAM: 4 GB

Por último se ha utilizado una conexión de Internet entre los dos ordenadores empleados en los experimentos (cable Ethernet dentro de la misma subred de 100 MBit/s).

	DESCRIPCIÓN	UTILIZACIÓN	LOGO
Microsoft Visual Studio 2010	Entorno de desarrollo integrado para sistemas operativos Windows	Programador del servidor del controlador	
Hammer	Editor de mapas en 3D	Creación de mapas para cargarlos sobre el simulador OVVV	
Gnuplot	Programa de realización de gráficas	Mostrar los resultados de las pruebas realizadas	
Lyx	Programa para la edición de texto usando LaTex	Generación de memoria	
Steam	Plataforma de distribución digital	Lanzar el simulador multicámara	
Microsoft Visio 2013	Software de dibujo vectorial	Realización de los diferentes esquemas incluidos en la memoria	

Cuadro 5.1: Programas utilizados.

5.3. Utilización de memoria y CPU

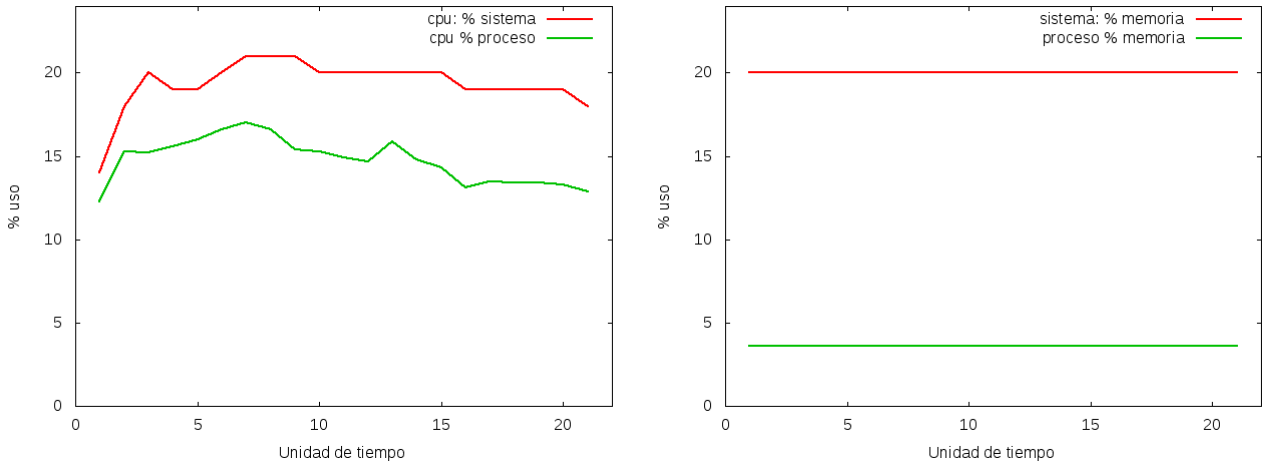
5.3.1. Descripción

Esta prueba consiste en mostrar los recursos de memoria y CPU que requiere tanto el Simulador Virtual como del Controlador. Las características donde se ejecutarán estos sistemas se encuentras definidos en la sección 5.2.

5.3.2. Condiciones del experimento

Para el desarrollo de estos experimentos lo que se ha realizado ha sido simulación recorriendo todas las posibilidades del simulador diseñado e ir midiendo los recursos que requiere del sistema y del proceso del simulador con respecto a la memoria y el uso de la CPU.

En las cuatro gráficas que se representán se muestrá en el eje 'y' la tasa de utilización y en el eje 'x' muestra el transcurso del tiempo por el cuál las aplicaciones utilizadas nos han geneado los resultados.



(a) Rendimiento CPU.

(b) Rendimiento memoria.

Figura 5.1: Memoria y CPU utilizada por el Simulador Virtual.

Para la realización de las pruebas en el Sistema Operativo Linux se llevo acabo utilizando la salida del comando ' top', y depués sobre esa salida se pasará un scrip selecionando unicamente los valores que se desee, y para la generación de pruebas en el Sistema Operativo Windows se utilizará la herramienta: Administrador de tareas. Por lo tanto, el Sistema Operativo Linux mide la memoria y porcentaje de CPU requerida por el Controlador, y sobre Windows se realizarán las mediciones con respecto a la Interfaz de Comunicaciones.

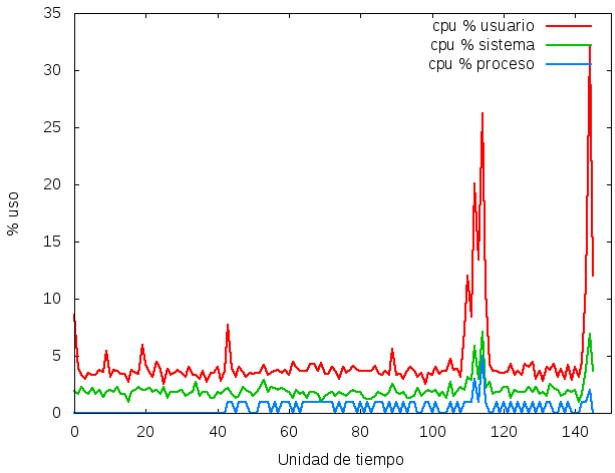
5.3.3. Resultados

En las Figuras 5.1 y 5.2 se muestran las gráficas que representan el porcentaje de utilización de memoria y CPU que requiere el Controlador y el Simulador Virtual.

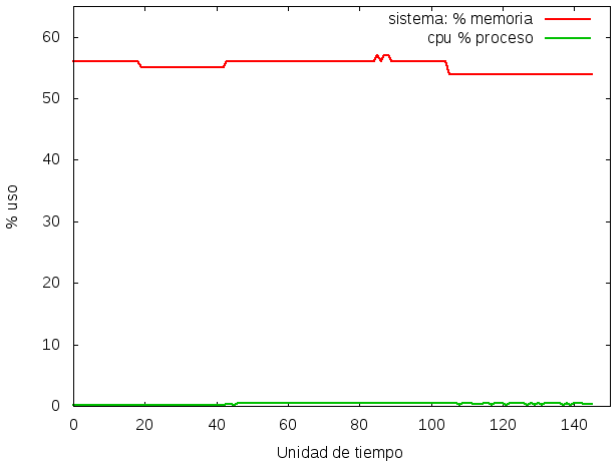
En figura 5.1a se muestra como el Simulador Virtual requiere casi la totalidad de los recursos de la CPU, de otro modo, en la figura 5.1b se muestra un bajo coste de recursos de memoria, teniendo un comportamiento bastante constante.

Con respecto al Controlador,el rendimiento de la CPU (figura 5.2) es muy bajo comparándolo con el que requiere el sistema, y con respecto al uso de la memoria, se observa en la figura 5.2b como hay una gran diferencia entre la que se encuentra utilizando en el sistema con respecto a la requerida por el Controlador, tiendo esta un bajo consumo.

Comparando los recursos que utiliza el Simulador Virtual con el Controlador, es el Simulador Virtual quien tiene mayor gasto de CPU, y en cuanto la utilización de la memoria, ambos tienen un comportamiento bastante lineal y con una baja tasa de uso con respecto a la que se encuentra utilizando en el sistema.



(a) Rendimiento CPU



(b) Rendimiento memoria.

Figura 5.2: Memoria y CPU utilizada por el Controlador.

5.4. Rendimiento de representación de imágenes

5.4.1. Descripción

Esta prueba mide la relación que existe entre el porcentaje de imágenes que se representarán de forma completa con respecto el tamaño de las imágenes variando la velocidad en que se recibirá en el Controlador, es decir, modificando el fps. Las imágenes que se han estudiado tienen los siguientes tamaños: 200x200, 300x300, 500x500 y 800x800.

5.4.2. Condiciones del experimento

Las mediciones de la prueba han sido tomadas en el Controlador, concretamente lo que se realizará es comparar el tamaño de la imagen recibido con el que deberia de ser, en el caso de que no coincidan, se marcára como error y se pasará a recibir la siguiente imagen.

Los valores que se representarán en los ejes de coordenadas de la gráfica son:

- % Error: Hace referencia a la tasa de imagenes que han llegado al Controlador estando incompletas.
- FPS: Se encuentra representado en mili-segundos, y muestra el tiempo de representación de la imagenes, de tal modo que a medida que aumente, la cantidad de imagenes que solicitará a la Interfaz de Comunicación será menor.

Para la medición de las pruebas se ha desarrollado sobre las primeros 500 imágenes que recibe el Controlador,

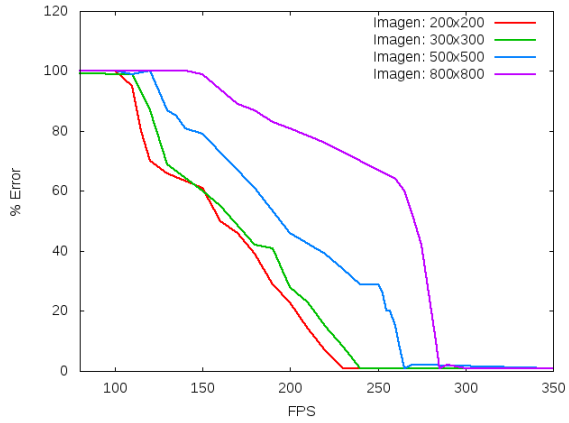


Figura 5.3: Tasa de error variando fps con respecto al tamaño de las imagenes.

5.4.3. Resultados

En las siguiente figura 5.3 se muestra el resultado de las pruebas.

De la figura 5.3 se observa que por debajo de los 100 mili segundos de un imagen , todas las imágenes llegarán incompletas, por lo que su tasa de error será del 100 %, a medida que se vaya aumentando el tiempo de mostrar las imágenes, la tasa de error ira decreciendo, sucediendo esta mejora mas rápido para las imágenes con menor tamaño.

Además, se ha analizado la tasa de error para sistemas con bajo indice de fallo, y se ha concluido de los análisis que la mayor parte de los errores que sufren en estos casos vienen dado en las primeras 10 imagenes que se reciban.

5.5. Tiempo de modificar una imagen en el sub-sistema Interfaz de Comunicación

5.5.1. Descripción

El objetivo de esta prueba es hacer un estudio sobre el tiempo que tarda en ser modificada una imagen desde que lo solicita la Interfaz de Comunicación hasta que el Motor Gráfico lo realiza. Para reproducir esta prueba el Controlador enviará solicitudes de mover la cámara con respecto a los siguientes movimientos (En la sección B.1 se muestra una gráfica del eje de coordenadas con respecto a los siguientes valores: X , Y, Z, RX, RY y RZ). En la Figura 5.4 todos los movimiento que permite simular, se encontrarán los movimientos ordenados, representando cada fila un movimiento distinto, en el orden en que se muestra es: X, Y, Z, RX, RY y RZ

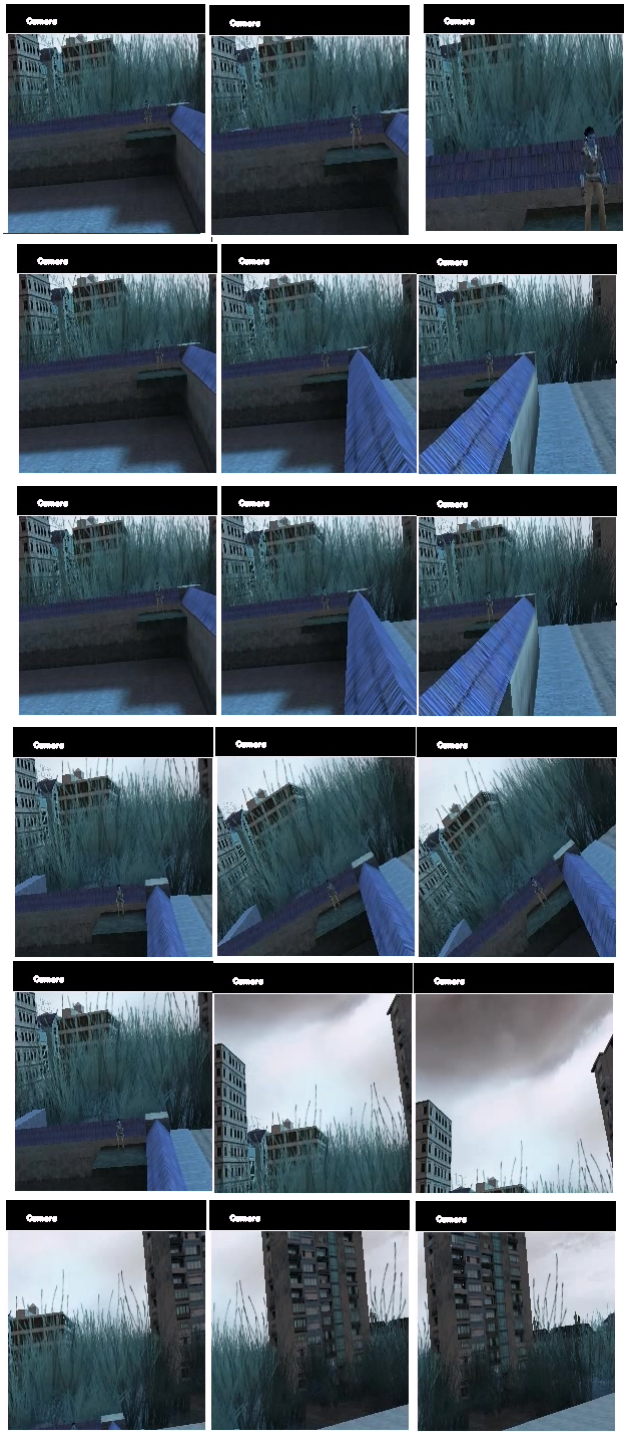


Figure 5.4: Movimiento de la cámara.

5.5.2. Condicciones del experimento

Se ha desarrollado el experimiento con una transmisón de frames de forma continuada desde el Controlador al entorno virtual pasando por la Interfaz de simulación. Las pruebas han sido llevadas acabo sobre el tamaño de imagen: 200x200. En cada una de las iterraciónes para recibir un frame se habrá hecho siempre una modificación cámara con respecto a su posicionamiento en el entorno virtual. A continuación se detalla el funcionamiento que llevará acabo la Interfaz de Comunicación para la captura de imágenes del entorno virtual en el orden en que se llevan acabo:

1. Modificar cámara: Realizará una petición al Motor Gráfico para modificar la posición de la cámara que se encuentra situada en el entorno virtual.

2. Captura de imágen: La Interfaz de Comunicación solicita una imagen de una determinada cámara.

3. Envio imagen: Una vez que la Interfaz de Comunicación obtenga la imagen devuelta por el Motor gráfico, se la enviará al Controlador. En este punto se habrá terminado una iterración y se volverá al punto 1.

El tiempo será medido en milisegundos, y se representán los siguientes tiempos en la gráfica; tiempo modificar y tiempo total. A continuación se realiza una explicación de cada uno de ellos:

- Tiempo modificar: Hace referencia al tiempo que transcure desde que la Interfaz de Simulación solicita mover la posición de la cámara, hasta que son movidos en el entorno virtual.

- Tiempo Total: Corresponde a la suma total de cada una de las operaciones que realiza la Interfaz de Comunicación por cada iterración. Se descompone de los siguientes tiempos:

  - Tiempo de modicar: Explicado en 5.5.2.
  - Tiempo capturar: Tiempo transcurrido desde que se la Interfaz de Simulación solicita una imagen hasta que el Motor devuelve la imagen del entorno virtual.
  - Tiempo envío: Tiempo que transcurre tarda en enviar la imagen desde la Interfaz de Simulación al controlador.

El último valor que que se representan en las gráficas será 'Iterración Frames', el cual se sumará en una unidad cada vez que la Interfaz realice todas las siguintes operaciones por cada imagen: Modificar cámara (sección 1) Captura de imagen (sección 2) y Envío de imágen (sección 3).

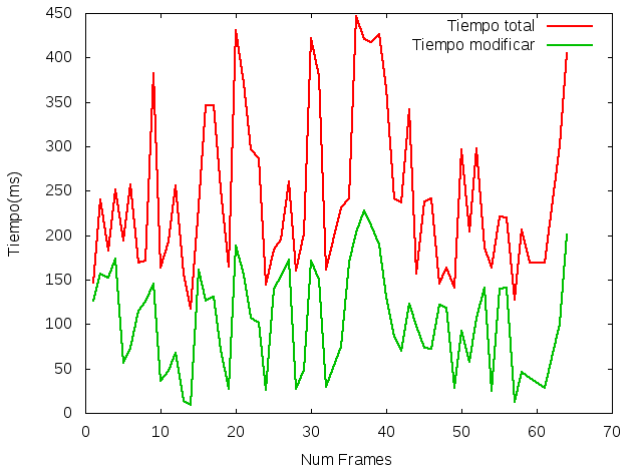


Figura 5.5: Gráfica medir tiempo de modifcar parámetros.

5.5.3. Resultados

En la Figura 5.5 se representan los resultados obtenidos que relacionan el Tiempo con cada iterración. Los tiempos comparados son tiempo total con tiempo de modificar.

De la gráfica mostrada en la Figura 5.5 se observa un comportamiento bastante sincronizado, es decir, cuando aumente uno de los dos tiempos, aumentará el otro y si uno disminuye, tambien diminuirá el otro. También se observa que tiende a ser la mitad el tiempo modificar con respecto al tiempo total.

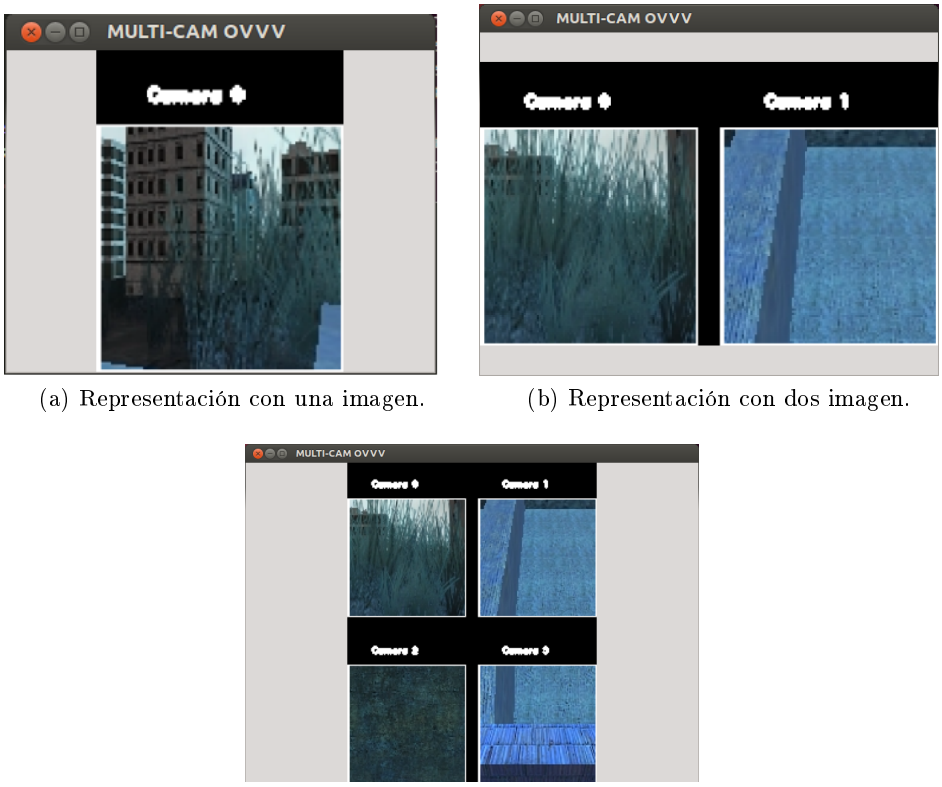
5.6. Rendimiento en el Controlador con respecto al número de imagenes.

5.6.1. Descripción

Las medicciones de esta prueba tienen lugar en el Controlador, y el objetivo de la prueba es ir variando el número de imagenes que se reciben de forma simultanea con respecto al tiempo que tarda en recibirlas. Las pruebas realizadas han sido llevadas acabo sobre 1, 2 y 4 imagenes recibidas a la vez de manera continuada. En la Figura 5.6 se muestra un ejemplo de como el Controlador muestra las imagenes.

5.6.2. Condiciones del experimento

Se han llevado acabo tres mediciones, cuya diferencia entre ellas es le numero de imágenes que recibe a la vez. El funcionamiento del Controlador con respecto a la funcionalidad de mostrar



(a) Representación con una imagen.

(b) Representación con dos imagen.

(c) Representación con cuatro imagen.

Figura 5.6: Interfaz del Controlador variando número de mostrar imágenes.

varias imagenes a la vez se explica a continuación:

1. Recibo de imagenes: El controlador recibirá una imagen por cada cámara, hasta que no termine no pasara al siguiente paso.

2. Mostrar imágen: Muestra todas las imagenes recibidas en el paso anterior.

3. Envio solicitud a la Interfaz: En este punto el Controlador podrá determinar seguir recibiendo imágenes, volviendo al primer punto, o podrá finalizar volviendo al menu principal.

De las tres anteriores operaciones, en esta prueba se estudiará la primera: Recibo de Imágenes variando el número de imágenes a recibir. Todas las cámaras tendrán la misma dimensión, 200x200, por lo que al tratarse de una imagen RGB, el tamaño de cada frame quedará definido de la siguiente manera:

- Una imágen: 120000 byts.
- Dos imágenes: 240000 byts.

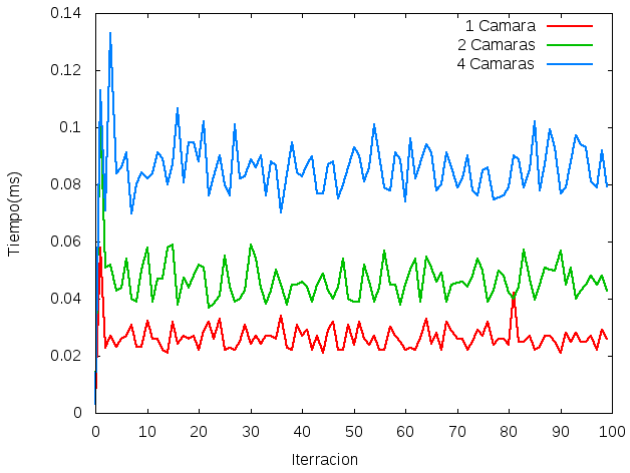


Figura 5.7: Gráfica de tiempo variando el numero de imágenes.

- Cuatro imágenes: 480000 byts.

La iterración que se representará en la sección resultados 5.6.3 hace referencia a cada ciclo fomado por el recibo de imágen, mostrar imágen y envío solicitud a la Interfaz.

5.6.3. Resultados

En la Figura 5.7 se muestra la representación gráfica de los resultados obtenidos para la simulacion sobre: una, dos y cuatro cámaras.

En la Figura 5.7 se obterva como los tres experimentos tienen un comporamiento muy constante, siendo aproximadamente el doble de tiempo que requierre en recibir dos imagenes que una, y cuatro imágenes que dos. Esta relación tiene que ver con respecto al tamaño de byts que recibe, ya que como se detallo en condiciones de experimientos es el doble (sección 5.6.2).

Por último, en las tres pruebas comienzan con un pico de tiempo y luego se estabilizan, este pico es una de las conclusiones de la prueba rendimiento de representación (sección 5.4).

Capítulo 6

Conclusión y trabajo futuro

6.1. Conclusiones

El sistema desarrollado permite controlar múltiples cámaras instaladas en entornos virtuales para la simulación de mapas en 3D. Además, este sistema permite desarrollar este control de manera remota, creando una separación entre controlador de las cámaras y el encargado de la creación-modificación de las cámaras en los diferentes escenarios.

De los objetivos establecidos 3.2, ha habido un requisito que no se ha podido llevar acabo, por lo que quedará pendiente para un trabajo futuro 6.2,el cual es adaptar el sistema creado al simulador de redes Wise 2.1.2 para el estudio de algoritmos de seguimiento y detección. El motivo por el cual no se ha podido llevar acabo es que ha habido dos requerimientos que han supuesto mayor esfuerzo de lo planeado generando un retraso en el progreso de proyecto, uno de ellos fue proceso el descifrado de los datos que devolvía el simulador cuando se le requería una imagen para transformarla a un formato visible, y el otro fue el manejo de envío de imágenes de múltiples cámaras de manera continuada.

6.2. Trabajo futuro

Se ha realizado un gran labor de desarrollo del sistema de simulación, por lo que ahora faltaría la fase de utilizar los datos que suministra para el estudio y mejora de algoritmos de seguimiento y detección. Para ello habría que adaptar el sistema creado al simulador de redes Wise. Para ello, durante el desarrollo del sistema del Controlador sé instauraría en un futuro Wise, ha sido desarrollado con esta intención, por lo que la adaptación no se estima requiera de un gran esfuerzo. Además, se podrá añadir a las funcionalidades la utilización de generación automática de ground truth, la cuál es una carácteristica que aporta OVVV y servirá de gran ayudar a la hora de definir la salida ideal del algoritmo analizado sobre las imágenes.

Bibliografía

[1] Object Video Inc. Object video virtual tool reference manual. http://development.objectvideo.com/ 2006. v, 27

[2] Juan C. SanMiguel, Jesús Bescós, José M. Martínez, and Álvaro García. DiVA: A distributed video analysis framework applied to video-surveillance systems. *WIAMIS '08. Ninth International Workshop on Image Analysis for Multimedia Interactive Services. IEEE,*, pages 207–210, Mayo 2008. 1

[3] Juan C. San Miguel, Christian Micheloni, Karen Shoop, Gian Luca Foresti, and Andrea Cavallaro. Self-reconfigurable smart camera networks. *Computer*, 47(5):67–73, Mayo 2014. 2

[4] Geoffrey R. Taylor, Andrew J.Chosak, and Paul C. Brewer. Ovvv: Using virtual worls to design and evaluate surveillanve systems. *Computer Vision and Pattern Recognition and 2007. CVPR '07. IEEE Conference on*, pages 1–8, Junio 2007. 5, 6

[5] Faisal Qureshi and Wiktor Starzyk. Software laboratory for camera networks research. *Emerging and Selected Topics in Circuits and Systems and IEEE Journal on*, 3(2):284–293, Abril 2007. 6, 7

[6] Lukas Esterle, Peter R. Lewis, Horatio Caine, Xin Yao, and Bernhard Rinner. Camsim: A distributed smart camera network simulator. *Self-Adaptation and Self-Organizing Systems Workshops (SASOW) and 2013 IEEE 7th International Conference on*, pages 19–20, Septiembre 2013. 7, 8

[7] C. Nastasi and A. Cavallaro. Wise-mnet: an experimental environment for wireless multimedia sensor networks. *Proc. of Sensor Signal Processing for Defence (SSPD)*, Septiembre 2011. 8, 9

[8] A. Pham and C.Makhoul. Performance study of multiple cover-set strategies for mission-critical video surveillance with wireless video sensors. *IEEE Int. Conf. on Wireless and Mobile Computing and Networking and Communications*, pages 208–216, Octubre 2010. 8, 9

[9] Torsten Braun Eduardo Cerqueira Hongli Xu Liusheng Huang Roger Immich Marilia Curado Zhongliang Zhao, Denis Rosario. M3wsn research project summary. *Institute of Computer Science and Applied Mathematics*, 2013. 9, 10

[10] Juan Carlos San Miguel Avedillo. Wise-mnet (wireless multimedia sensor networks) overview and hands-on. *Quen Mary. Universidad de Londres*, Mayo 2014. 11

Apéndice A

Manual de uso

A continuación se detalla un manual de uso que describe la utilización de la herramienta creada.La primera herramienta que se abrirá será la encargada de simular el escenario virtual, Source SDK 2.2.1, para ello se requierre el programa Steam. Para ello se pulsará sobre el icono del programa (vease tabla ..) , y una vez logeado aparecerá la siguiente pantalla A.1.

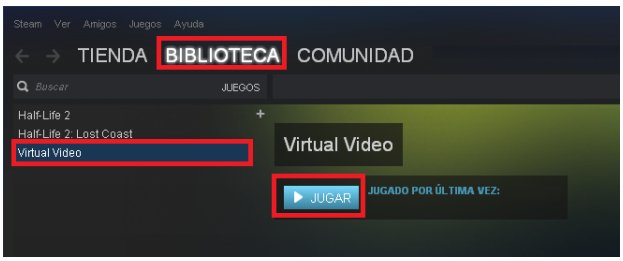


Figure A.1: Manual de uso - Virtual Video: paso 1.

Se pulsará sobre: BIBLIOTECA -> Virtual Video -> Jugar. La siguiente pantalla A.2 que aparecerá será la sigueinte.

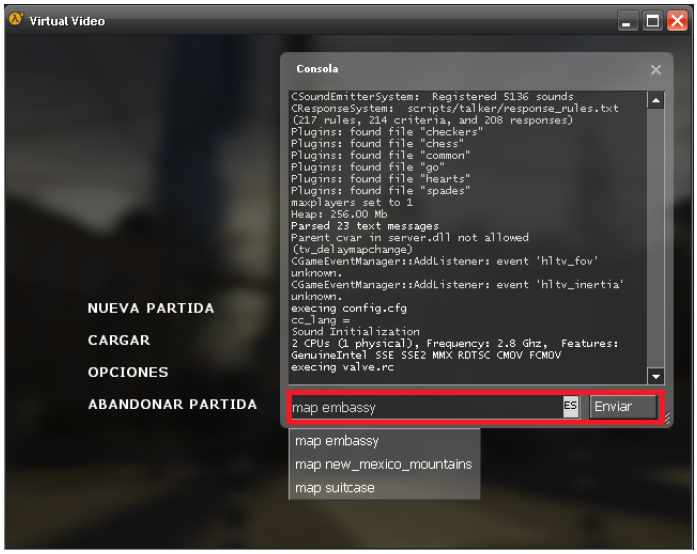


Figure A.2: Manual de uso - Virtual Video: paso 2.

Sobre la consola se elegirá el mapa 3D que se quierra cargar, y despues se pulsará a enviar. Por último, Virtual Video presenta una pantalla A.3 donde muestra el escenario que se ha cargado.

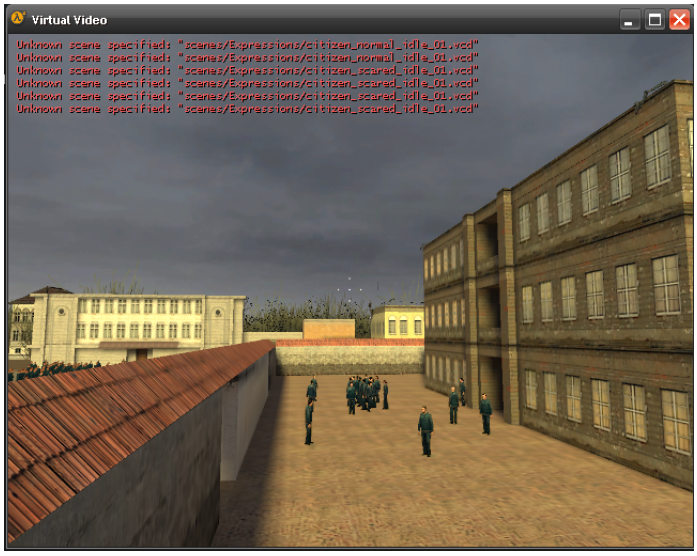


Figure A.3: Manual de uso - Virtual Video: paso 3.

La siguiente herramienta que se abrirá será la encargada del manejo de las cámaras. En los siguientes pasos se explica como abrir el proyecto y ejecutarlo. Para ello, primero se abrirá la herramienta: Visual Studio 2010, apareciendo la siguiente pantalla A.4.

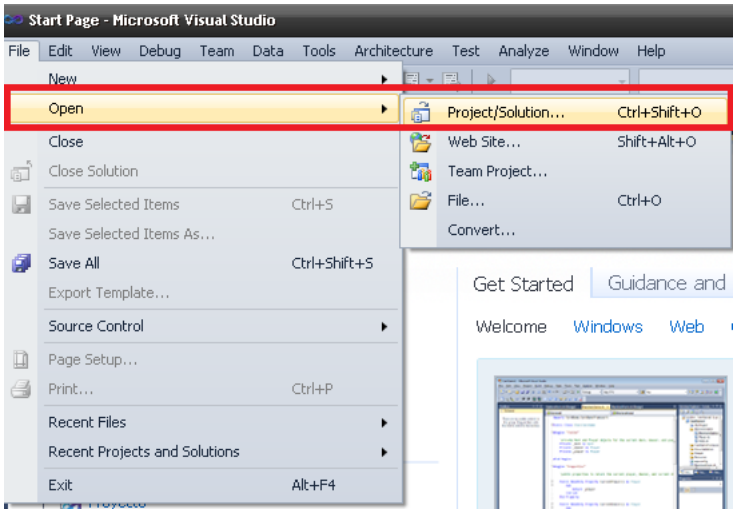


Figure A.4: Manual de uso - Simulador de cámaras: paso 1.

En el margen superior izquierdo se pulsará sobre el botones: File -> Open -> Project/Solution. Y aparecerá la siguiente pantalla A.5.

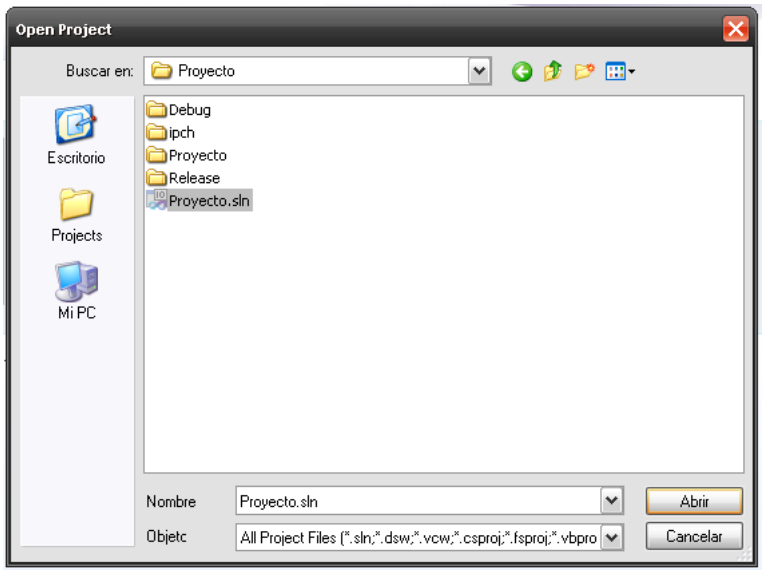


Figure A.5: Manual de uso - Virtual Video: paso 1.

Por último, se pulsará sobre el botón marcado en rojo para arrancar el simulador de cámaras.

Figure A.6: Manual de uso - Simulador de cámaras: paso 3.

La última herramienta que se abrirá será el controlador, el cual se encuentra desarrollado para el sistema operativo Linux. El primer paso será situarnos con la terminal sobre la carpeta que contenga los ficheros y realizar *make*.

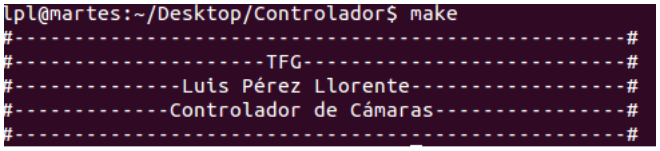


Figure A.7: Manual de uso - Controlador: paso 1.

Para ejecutar el programa se escribirá: *./Controlador*

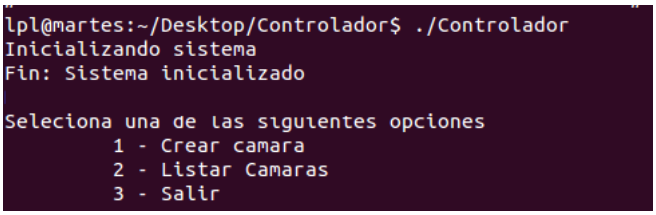


Figure A.8: Manual de uso - Controlador: paso 2.

En este punto se encontrará el ususario en el Menu Principal. Si se elige crear una cámara la pantalla que aparecerá sera la siguiente A.9. En ella habra qur ir introduciendo los diferentes valores para los campos que conforma cada cámara.

```
Introduza valor de FOV:
40
Introduza valor de ancho:
450
Introduza valor de alto:
250
Numero frames:
15
Camara trasnlation: x
84
Camara trasnlation: y
0
Camara trasnlation: z
9
Camara trasnlation: rx
12
Camara trasnlation: ry
0
Camara trasnlation: rz
0
```

Figure A.9: Manual de uso - Controlador: paso 3.

Por último se muestra un ejemplo de como el sistema muestra las imágenes de las cámaras.

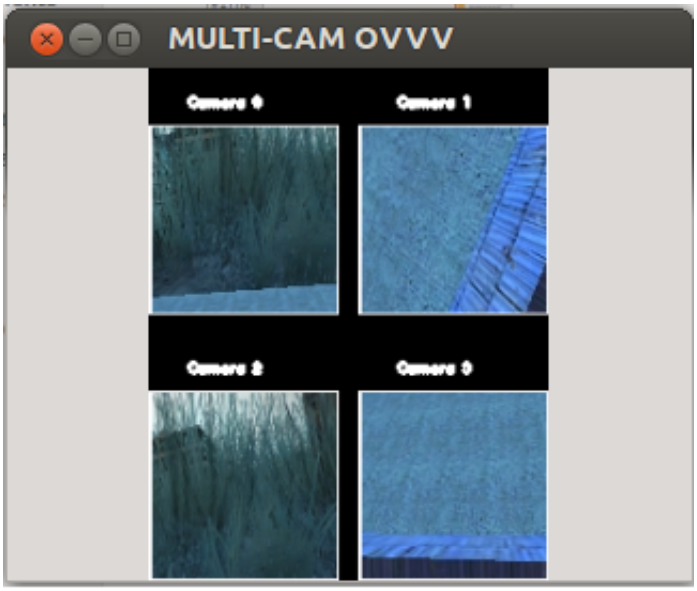


Figure A.10: Manual de uso - Controlador: paso 4.

Apéndice B

Librerías

En esta sección se presenta las tres librerías principales que se utilizarán. La librería Socket se utiliza para el desarrollo de la comunicación entre los sistemas, mientras que la librería Virtual Video C se utiliza para el manejo de las cámara, por último , la librería OpenCV se utiliza para mostrar las imágenes captadas por la cámaras.

B.1. Librería Virtual Video C:

En el capítulo de análisis y diseño, en el apartado de esta librería **??**, se realizó una prestación de la arquitectura, en la que explicó como el simulador de cámaras se encontraba formado por dos servidores: Servidor de cámara y Servidor PTZ. Para la realización de este proyecto solo se requiere de la utilización de algunas partes del Servidor de cámara, por lo que a partir de ahora no se explicará las caracteristicas que conforma el Servidor PTZ, centrandose únicamente en los campos que se utilizan del Servidor de cámara. A continuación se especifican se detallan los campos utilizados del Servidor de cámara:

- Socket: Único por cada cámara, servirá para el protocolo de comunicación.
- Identificador: Se utilizará para distinguir cada cámara del resto.
- Tipo: Cámara proyectiva
- Tamaño
    - Ancho: Almacena la dimensión del ancho de la imagen.
    - Alto: Almacena la dimensión del alto de la imagen.
- Framerate: Almacena el número de frames por segundo.
- FOV: Campo de visión.

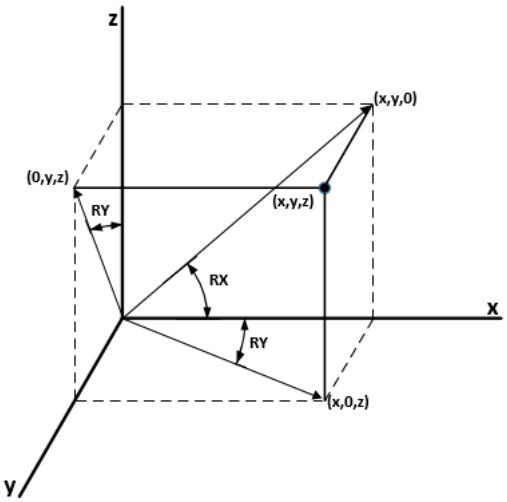


Figura B.1: Representation gráfica de un punto.

- Posicionamiento: Los valores que determina la posición en un plano son los siguiente; x, y , z, - Rx, Ry, Rz. En la FiguraB.1 se muestra una representación de un punto con respecto a los anteriores variables.
- FrameNum: Número de imagen seleccionada.
- Formato: 24 bits RGB.
- Imagen:
  - Tamaño: Contiene el tamaño en byts de la imagen.
  - Datos: contiene la última imagen capturada.

La librerías que se requiere para su utilización son: *<VVClient.h>*

B.2. Socket

Para la realización del trabajo fin de grado se ha requerido realizar dos conexiones tal y como se muestra en la Figura 4.3. Todas ellas seguirán el mismo esquema, TCP/IP. En el apartado conexiones 4.4, se muestra las funciones utilizadas de esta librería.

La librerías que se requiere para su utilización son: *<sys/types.h>* y *<sys/socket.h>*.

B.3. OpenCV

Esta librería sera utilizada para mostrar las imágenes capturas por las cámaras. Como se muestra en la imagen 3.4, este modulo es llamado en dos ocasiones en el Controlador, para mostrar una única cámara o para mostrar múltiples cámaras. En el aparato **??** se muestra el pseudocodigo de la función de recibir-mostrar una imagen.

La librerías que se requiere para su utilización son: *<opencv2/core/core.hpp>* y *<opencv2 /highgui/highgui.hpp>*.